



**ITS**  
Institut  
Teknologi  
Sepuluh Nopember

**TUGAS AKHIR - KI1502**

# **MEMPERBAIKI BUSINESS PROCESS MENGANDUNG INVISIBLE TASK MENGGUNAKAN GRAPH MODEL**

**REYNALDO JOHANES**  
**NRP 5114 100 064**

**Dosen Pembimbing I**  
**Prof. Drs. Ec. Ir. Riyanarto Sarno, M.Sc.,Ph.D**

**Dosen Pembimbing II**  
**Dwi Sunaryono, S.Kom.,M.Kom.**

**DEPARTEMEN INFORMATIKA**  
**Fakultas Teknologi Informasidan Komunikasi**  
**Institut Teknologi Sepuluh Nopember**  
**Surabaya 2018**

*[Halaman ini sengaja dikosongkan]*

**TUGAS AKHIR - KI1502**

# **MEMPERBAIKI BUSINESS PROCESS MENGANDUNG INVISIBLE TASK MENGGUNAKAN GRAPH MODEL**

**REYNALDO JOHANES**  
**NRP 5114 100 064**

**Dosen Pembimbing I**  
**Prof. Drs. Ec. Ir. Riyanarto Sarno, M.Sc.,Ph.D.**

**Dosen Pembimbing II**  
**Dwi Sunaryono, S.Kom.,M.Kom.**

**DEPARTEMEN INFORMATIKA**  
**Fakultas Teknologi Informasi dan Komunikasi**  
**Institut Teknologi Sepuluh Nopember**  
**Surabaya 2018**

*[Halaman ini sengaja dikosongkan]*

**FINAL PROJECT - KI1502**

# **BUSINESS PROCESS REFINEMENT CONTAINING INVISIBLE TASK USING GRAPH MODEL**

**REYNALDO JOHANES**  
**NRP 5114 100 064**

**Supervisor I**  
**Prof. Drs. Ec. Ir. Riyanarto Sarno, M.Sc., Ph.D.**

**Supervisor II**  
**Dwi Sunaryono, S.Kom., M.Kom.**

**DEPARTMENT OF INFORMATICS**  
**Faculty of Information and Communication Technology**  
**Institut Teknologi Sepuluh Nopember**  
**Surabaya 2018**

*[Halaman ini sengaja dikosongkan]*

## LEMBAR PENGESAHAN

### MEMPERBAIKI BUSINESS PROCESS MENGANDUNG INVISIBLE TASK MENGGUNAKAN GRAPH MODEL

#### TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat  
Memperoleh Gelar Sarjana Komputer  
pada  
Rumpun Mata Kuliah Manajemen Informasi  
Sistem Studi S-1 Departemen Informatika  
Fakultas Teknologi Informasi dan Komunikasi  
Institut Teknologi Sepuluh Nopember

Oleh

**REYNALDO JOHANES**

NRP. 5114 100 064

Disetujui oleh Dosen Pembimbing Tugas Akhir:

Prof. Drs. Ec. Ir. RIYANARTO  
M.Sc., Ph.D.  
NIP: 19590803 198601 1 001

DWI SUNARYONO,  
S.Kom., M.Kom.  
NIP: 198608232015041004



**SURABAYA  
JANUARI, 2018**

*[Halaman ini sengaja dikosongkan]*



# MEMPERBAIKI BUSINESS PROCESS MENGANDUNG INVISIBLE TASK MENGGUNAKAN GRAPH MODEL

**Nama** : Reynaldo Johaness  
**NRP** : 5114100064  
**Departemen** : Informatika – FTIK ITS  
**Dosen Pembimbing I** : Prof. Drs. Ec. Ir. Rianarto Sarno,  
M.Sc.,Ph.D.  
**Dosen Pembimbing II** : Dwi Sunaryono, S.Kom.,M.Kom.

## Abstrak

Event Log merupakan salah satu aset penting di perusahaan. Penelitian ini berusaha untuk memodelkan model proses bisnis di event log sehingga bisa menganalisa proses model agar bisa membawa berbagai manfaat bagi perusahaan seperti mengurangi biaya perusahaan, membawa keuntungan meningkat, dan banyak lagi. Salah satu masalah dalam pemodelan *event log* secara otomatis adalah penambahan *invisible task*. Tugas akhir ini bertujuan untuk mengembangkan sebuah sistem yang dapat memodelkan proses bisnis dari *event log* yang berisi *invisible task* dengan menggunakan graph database di *Neo4J*. Pertama-tama, penelitian ini menghasilkan event log into a link list berupa graph di *Neo4J*. Selanjutnya, tugas akhir ini mengusulkan sebuah metode untuk meningkatkan link list dari *event log* menjadi model graph yang mengandung *invisible task*. Hasil model graph yang diperoleh dengan menggunakan *Neo4J* dibandingkan dengan hasil yang diperoleh dengan menggunakan LTL yang diturunkan dari aturan dalam model deklaratif. Tugas akhir ini dapat membuktikan bahwa metode yang diusulkan dengan menggunakan graph database adalah metode yang tepat karena hasil graph memiliki hasil yang lebih baik daripada *Heuristic Miner*. Hasil *Neo4J* bisa langsung menghasilkan relasi yang benar meski event log mengandung invisible task dimana tidak bisa dilakukan pada *Heuristic Miner*.

**Kata kunci:** *ControlFlow Pattern, Invisible Task, Neo4J, Graph Database*

*[Halaman ini sengaja dikosongkan]*

## **BUSINESS PROCESS REFINEMENT CONTAINING INVISIBLE TASK USING GRAPH MODEL**

**Student Name** : Reynaldo Johaness  
**NRP** : 5114100064  
**Major** : Informatics – FTIK ITS  
**Supervisor I** : Prof. Drs. Ec. Ir. Riyanarto Sarno,  
M.Sc.,Ph.D.  
**Supervisor II** : Dwi Sunaryono, S.Kom.,M.Kom.

### ***Abstract***

Event log is one of the important assets in the company. This research strive to model the business process model in event log so we can analyze the model process in order to bring various benefits to companies such as reducing company costs, increasing profits, and more. One of the problem in modeling event log automatically is the addition of invisible tasks. This research aims to develop a system that can model a business processes of an event log that contains invisible tasks using a graph database in Neo4J. First of all, this researchmodels the event log into a link list in the form of graph in Neo4J. Next,this research proposes a method to enhance the link list of event log to be a graph model that contains the invisible task. The results of the obtained graph model using Neo4Jare compared with the results that are obtained using LTL that is derived from rules in declarative model. This research can prove that the proposed method using graph database is a right method because the graph result have a better result than a Heuristic Miner. The result of Neo4J can directly generate relation which is true even though event log containing invisible task where can't be done in Heuristic Miner.

**Keywords:***ControlFlow Pattern, Invisible Task, Neo4J, Graph Database*

*[Halaman ini sengaja dikosongkan]*

## KATA PENGANTAR

Segala puji syukur penulis kepada Tuhan Yesus Kristus atas penyertaan-Nya, sehingga tugas akhir berjudul “Memperbaiki Business Process Mengandung Invisible Task Menggunakan Graph Model” ini dapat selesai sesuai dengan waktu yang telah ditentukan.

Pengerjaan tugas akhir ini menjadi sebuah sarana untuk penulis memperdalam ilmu yang telah didapatkan di Institut Teknologi Sepuluh Nopember Surabaya, khususnya dalam disiplin ilmu Teknik Informatika. terselesaikannya buku tugas akhir ini tidak terlepas dari bantuan dan dukungan semua pihak. Pada kesempatan kali ini penulis ingin mengucapkan terima kasih kepada:

1. Mama, papadan keluarga yang selalu memberikan dukungan berupa doa dan nutrisiselama proses pengerjaan Tugas Akhir.
2. Bapak Riyanarto Sarno dan Bapak Dwiselaku dosen pembimbing yang telah bersedia meluangkan waktu selama proses pengerjaan Tugas Akhir.
3. Bapak dan Ibu dosen Jurusan Teknik Informatika ITS yang banyak memberikan ilmu dan bimbingan bagi penulis.
4. Teman-teman RMP yang selalu mendukung dan menyemangati saya.
5. Teman-teman TC Angkatan 2014 yang selalu mendukungselama proses pengerjaan tugas akhir.
6. Kelly Rossa, Dewi Rahmawati yang membantu penulis dalam pengerjaan tugas akhir.
7. Serta semua pihak yang turut membantu penulis dalam menyelesaikan tugas akhir ini.

Penulis menyadari bahwa tugas akhir ini masih memiliki banyak sekali kekurangan. Dengan kerendahan hati, penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan ke depan.

Surabaya, Januari 2018

*[Halaman ini sengaja dikosongkan]*

## DAFTAR ISI

<b>LEMBAR PENGESAHAN.....</b>	<b>Error! Bookmark not defined.</b>
<b>Abstrak.....</b>	<b>ix</b>
<b><i>Abstract</i> .....</b>	<b>xi</b>
<b>DAFTAR ISI.....</b>	<b>xv</b>
<b>DAFTAR GAMBAR .....</b>	<b>xviii</b>
<b>DAFTAR TABEL.....</b>	<b>xxi</b>
<b>DAFTAR KODE SUMBER.....</b>	<b>xxii</b>
<b>DAFTAR SINGKATAN .....</b>	<b>xxiv</b>
<b>1      BAB I PENDAHULUAN .....</b>	<b>1</b>
1.1      Latar Belakang .....	1
1.2      Rumusan Permasalahan.....	2
1.3      Batasan Permasalahan .....	2
1.4      Tujuan.....	3
1.5      Manfaat.....	3
1.6      Metodologi .....	3
1.7      Sistematika Penulisan.....	5
<b>2      BAB II DASAR TEORI.....</b>	<b>7</b>
2.1 <i>Process Discovery</i> .....	7
2.2 <i>Streaming Event Log</i> .....	8
2.3 <i>Declarative Miner</i> .....	9
2.4 <i>NEO4J</i> .....	11
2.5 <i>Invisible Task</i> .....	12
2.6      Linear Temporal Logic (LTL).....	13
2.7 <i>Graph Database</i> .....	14

<b>3</b>	<b>BAB III METODE PEMECAHAN MASALAH.....</b>	<b>16</b>
3.1	Cakupan Permasalahan.....	17
3.2	Masukan .....	18
3.2.1	Data.....	18
3.3	Pre-processing .....	19
3.3.1	Transformasi ke Log Data berdasarkan Database.....	19
3.3.2	Filtering List Aktivitas pada Log Data.....	24
3.3.3	Pemisahan Log Data .....	24
3.4	Pembentukan <i>Business Process Model</i> .....	25
3.4.1	Pembentukan <i>Business Process Model</i> dari Model Deklaratif .....	25
3.4.2	Pembentukan <i>Business Process Model</i> dari Model Neo4J .....	30
3.4.3	Pembentukan <i>Business Process Model</i> dari Model Heuristic Miner .....	32
3.5	Keluaran .....	33
3.5.1	Metode Pengujian .....	33
3.5.2	Skenario Uji Coba .....	33
<b>4</b>	<b>BAB IV IMPLEMENTASI.....</b>	<b>34</b>
4.1	Lingkungan Implementasi .....	35
4.1.1.	Perangkat Keras.....	35
4.1.2.	Perangkat Lunak .....	35
4.2	Penjelasan Implementasi .....	36
4.2.1	Implementasi .....	36
<b>5</b>	<b>BAB V PENGUJIAN DAN EVALUASI .....</b>	<b>54</b>
5.1	Lingkungan Uji Coba .....	55



5.2	Pre-Processing .....	55
5.2.1	Hasil Transformasi ke Log Data .....	55
5.2.2	Hasil Pemisahan Message dari Log Data serta Pengelompokan Data per Bulan .....	56
5.3	Hasil Implementasi Pembentukan Business Process Model .....	57
5.3.1	Hasil Implementasi Pembentukan Business Process Model dari Model Deklaratif.....	58
5.3.2	Hasil Implementasi Pembentukan Business Process Model dari Model Neo4J .....	63
5.3.3	Hasil Implementasi Pembentukan Business Process Model dari Model Heuristic Miner.....	66
5.4	Hasil Perbandingan 3 Graph.....	68
<b>6</b>	<b>BAB VI KESIMPULAN DAN SARAN .....</b>	<b>69</b>
6.1	Kesimpulan.....	69
6.2	Saran.....	69
	<b>DAFTAR PUSTAKA .....</b>	<b>71</b>
	<b>LAMPIRAN .....</b>	<b>73</b>
	<b>DAFTAR ISTILAH.....</b>	<b>74</b>
	<b>BIODATA PENULIS .....</b>	<b>77</b>

*[Halaman ini sengaja dikosongkan]*

## DAFTAR GAMBAR

Gambar 2.1 Contoh Model Proses.....	8
Gambar 2.2 Definisi <i>Event Log</i> .....	9
Gambar 2.3. Contoh Node dalam Neo4J .....	11
Gambar 2.4. Contoh Relationship dalam Neo4J .....	12
Gambar 2.5 Contoh Invisible Task.....	13
Gambar 2.6. Operator Modal Temporal (LTL) .....	14
Gambar 2.7. Stuktur Data Graph Database .....	15
Gambar 3.1. Alur Proses Pengerjaan Tugas Akhir.....	18
Gambar 3.2. Pseudocode Pengisian Detail Attachment ...	24
Gambar 3.3. Pseudocode pemisahan aktivitas dan message	24
Gambar 3.4. Alur Hirarki Pembentukan Control-Flow Patterns .....	25
Gambar 3.5. Memasukkan event log ke Neo4J .....	30
Gambar 4.1. Memasukkan Event Log ke Disco.....	52
Gambar 4.2. Memasukkan Event Log ke ProM .....	52
Gambar 4.3. Heuristic Miner.....	53
Gambar 4.4. Memasukkan nilai untuk melakukan Heuristic Miner.....	53
Gambar 5.1 Database Proses Impor Barang Terminal Peti Kemas .....	56
Gambar 5.2 Log Data Proses Impor Barang Terminal Peti Kemas .....	57
Gambar 5.3 Log Data Tanpa <i>Message</i> pada Proses Impor Barang Terminal Peti Kemas.....	57
Gambar 5.4. Hasil Graph dari LTL .....	61
Gambar 5.5. Hasil Switch dari LTL .....	62
Gambar 5.6. Hasil Redo dari LTL.....	63
Gambar 5.7. Hasil Link List.....	63
Gambar 5.8. Hasil Graph dari Neo4J .....	64
Gambar 5.9. Hasil Link List Switch.....	64
Gambar 5.10. Hasil Link List Redo.....	65
Gambar 5.11. Hasil Switch langsung dari Event Log .....	65
Gambar 5.12. Hasil Redo langsung dari Event Log .....	65
Gambar 5.13. Hasil Graph dari Heuristic Miner .....	66

Gambar 5.14. Hasil Switch dari Heuristic Miner .....	67
Gambar 5.15. Hasil Redo dari Heuristic Miner.....	68

## DAFTAR TABEL

Tabel 2.1 Potongan <i>Template</i> Declare.....	10
Tabel 3.1. Cara Transformasi ke Log Data secara Garis Besar.....	19
Tabel 3.2. Perhitungan Waktu Aktivitas untuk Log Data	22
Tabel 3.3. Contoh Hasil Estimasi Waktu Setiap Aktivitas	23
Tabel 3.4. Metode untuk membangun control-flow-patterns.....	26
Tabel 3.5. Mengubah LTL ke Neo4J.....	28
Tabel 3.6. Penemuan Pola dalam Neo4J .....	30
Tabel 5.1. Hasil LTL TPS .....	58
Tabel 5.2. Hasil LTL Switch .....	61
Tabel 5.3. Hasil LTL Redo.....	62

*[Halaman ini sengaja dikosongkan]*

## DAFTAR KODE SUMBER

Kode Sumber 4.1. Transformasi Data .....	37
Kode Sumber 4.2. Filter data dari Message.....	37
Kode Sumber 4.3. Mencari nilai total_after .....	39
Kode Sumber 4.4. Mencari nilai total_before .....	39
Kode Sumber 4.5. Mencari nilai total_after_resp.....	39
Kode Sumber 4.6. Mencari nilai total_before_resp.....	40
Kode Sumber 4.7. Potongan Declarative Miner Pada Tahap Kedua.....	41
Kode Sumber 4.8. Mencari nilai totalEXsplit .....	41
Kode Sumber 4.9. Mencari nilai totalCRsplit .....	42
Kode Sumber 4.10. Mencari nilai totalCRnext .....	42
Kode Sumber 4.11. Mencari nilai totalEXjoin .....	42
Kode Sumber 4.12. Mencari nilai totalCRjoin .....	43
Kode Sumber 4.13. Mencari nilai totalCRbef .....	43
Kode Sumber 4.14. Potongan Declarative Miner pada Tahap Keempat.....	45
Kode Sumber 4.15. Potongan Declarative Miner pada Tahap Kelima .....	46
Kode Sumber 4.16. Pengubahan LTL ke Format CSV pada Relasi LTLXORSPLIT .....	47
Kode Sumber 4.17. Pseudocode Pengubahan LTL ke Format CSV pada Relasi LTLXORJOIN .....	48
Kode Sumber 4.18. Pseudocode Pengubahan LTL ke Format CSV pada Relasi LTLSequence.....	49
Kode Sumber 4.19. Memodelkan Model Bisnis Proses dari LTL .....	50
Kode Sumber 4.20. Mengubah event log ke link list .....	50
Kode Sumber 4.21. Membentuk Relasi di Neo4J .....	51
Kode Sumber 4.22. Mengecek dan Memperbaiki Invisible Task .....	51

*[Halaman ini sengaja dikosongkan]*



## DAFTAR SINGKATAN

<i>LTL</i>	: <i>Linear Temporal Logic.</i>
<i>CSV</i>	: <i>Comma Separated Value.</i>
<i>ProM</i>	: <i>Process Mining.</i>

*[Halaman ini sengaja dikosongkan]*

# **BAB I**

## **PENDAHULUAN**

Pada bab ini akan dipaparkan mengenai garis besar Tugas Akhir yang meliputi latar belakang, tujuan, rumusan dan batasan permasalahan, metodologi pengerjaan Tugas Akhir, dan sistematika penulisan Tugas Akhir.

### **1.1 Latar Belakang**

Event log adalah penyimpanan informasi proses dalam sebuah sistem [1]. Informasi dasar yang disimpan oleh event log terdiri dari nama kegiatan, sumber daya yang melakukan aktivitas, dan waktu kegiatan. Potongan informasi tersebut digunakan dalam analisis proses dan penemuan pemecahan masalah [2]. Singkatnya, event log adalah bahan untuk menganalisa proses yang dijalankan dan menemukan masalah dalam proses.

Karena event log menyimpan banyak informasi, proses penemuan nampaknya memudahkan analisa proses. Proses penemuan mengatur informasi event log dalam model proses, sehingga pengguna dapat dengan mudah mengevaluasi prosesnya. Penemuan proses adalah salah satu bagian dari proses penambangan yang telah menangani isu-isu di banyak sektor: bisnis [3] - [6], penipuan [1], [7], medis [8], [9], dan iklan [10]. kita perlu mencari model proses bisnis dari event log.

Tapi, kita punya banyak masalah untuk memodelkan model proses bisnis dari event log seperti pemodelan event log yang mengandung invisible task. Kita tidak bisa memodelkan invisible task karena invisible task tidak tercatat dalam log peristiwa. Selain masalah di atas, kita tidak bisa memodelkan log peristiwa ke model proses bisnis secara manual karena sangat merugikan di bidang waktu dan usaha. Jika kita tidak menambah

kan sebuah invisible task maka kita akan bingung menentukan relasinya.

Dari masalah itu maka kita membutuhkan sebuah sistem yang bisa memodelkan secara otomatis dan bisa menangani event log yang mengandung invisible task. Kita mengusulkan menggunakan graph database untuk memodelkannya karena jika di relational database biasa tidak akan bisa sampai mengecek relasi antar tabel. Oleh karena itu, Tugas akhir ini bertujuan untuk mengembangkan sebuah sistem yang dapat memodelkan model proses bisnis dari event log yang mengandung invisible task dengan menggunakan graph database di Neo4J.

## 1.2 Rumusan Permasalahan

Rumusan masalah yang diangkat dalam Tugas Akhir ini dapat dipaparkan sebagai berikut:

1. Bagaimana cara mentransform data dari database menjadi event log?
2. Bagaimana cara membentuk rule *LTL* (*Linear Temporal Logic*) berdasarkan log data yang mengandung *invisible task* dengan menggunakan declarative miner?
3. Bagaimana cara membentuk graph model dari rule *LTL* (*Linear Temporal Logic*)?
4. Bagaimana cara membentuk graph model dan memperbaikinya dari event log yang mengandung invisible task di *Neo4J*?
5. Bagaimana cara membentuk graph model menggunakan *Heuristic Miner*?
6. Metode apakah yang merupakan metode yang tepat untuk menggambarkan *Business Model Process*.

## 1.3 Batasan Permasalahan

Permasalahan yang dibahas dalam Tugas Akhir ini memiliki beberapa batasan, di antaranya sebagai berikut:

1. Implementasi *Business Process* dengan database *Neo4j*.

2. Data masukan berupa *event log* dalam bentuk Excel yang diproses menggunakan *declarative miner* pada Java sehingga membentuk suatu model.
3. Data masukan berupa *event log* dalam bentuk Excel yang diproses menggunakan *Cypher* pada Neo4J sehingga membentuk suatu model.

## 1.4 Tujuan

Tujuan dari Tugas Akhir ini adalah untuk mendapatkan sebuah *graph model* yang dapat menangani *invisible task* yang didapatkan dari pembentukan *control-flow patterns* menggunakan *declarative miner*, *Neo4J* secara langsung, dan *heuristic miner*, sehingga kita dapat mengetahui metode mana yang terbaik untuk mendapatkan sebuah *graph model* yang dapat menangani *invisible task*.

## 1.5 Manfaat

Penelitian ini diharapkan dapat membentuk *graph model* yang berasal dari kumpulan *control-flow patterns* yang dibentuk dari model deklaratif dan log data. Namun *graph model* yang dibentuk telah menangani *invisible task*, sehingga data business proses lebih mudah untuk diolah.

## 1.6 Metodologi

Langkah-langkah yang ditempuh dalam pengerjaan Tugas Akhir ini yaitu sebagai berikut:

### 1. Studi literatur

Dalam pembuatan Tugas Akhir ini telah dipelajari tentang hal-hal yang dibutuhkan sebagai ilmu penunjang dalam penyelesaiannya. Ilmu penunjang utama pada Tugas Akhir ini adalah permodelan menggunakan Neo4j dan Declarative Miner. Selain itu,

terdapat literatur lain yang menunjang proses penyelesaian Tugas Akhir ini.

## **2. Pemodifikasian Metode**

Pada tahap ini penulis menjabarkan cara pemecahan masalah yang terdapat dalam rumusan masalah.

## **3. Analisis dan Perancangan Sistem**

Aktor yang menjadi pelaku adalah pengguna perangkat lunak yang dibangun oleh penulis. Kemudian beberapa kebutuhan fungsional dari sistem ini adalah sebagai berikut:

- a. Preprocessing data yang akan digunakan sebagai input.
- b. Menemukan rule LTL dari *event log* dengan menggunakan *Declarative Miner*.
- c. Memodelkan rule LTL yang didapatkan dari declarative miner ke bentuk graph menggunakan Neo4j.
- d. Memodelkandan memperbaiki relasi graph dari event log yang mengandung invisible task
- e. Memodelkan *event log* ke *graph model* menggunakan *Heuristic Miner*.

## **4. Implementasi**

Pada tahap ini dilakukan pembuatan perangkat lunak yang menghasilkan tiga graph dengan menggunakan neo4j yang terbentuk dari hasil *Declarative Miner*, input langsung dari *event log*, dan *Heuristic Miner*, kemudian membandingkan ketiga hasil tersebut.

## **5. Pengujian dan evaluasi**

Pada tahap ini dilakukan pengujian terhadap elemen perangkat lunak dengan menggunakan data uji yang telah dipersiapkan. Pengujian dan evaluasi perangkat lunak dilakukan untuk mengevaluasi

jalannya perangkat lunak, mengevaluasi fitur utama, mengevaluasi fitur-fitur tambahan, mencari kesalahan yang timbul pada saat perangkat lunak aktif, dan mengadakan perbaikan jika ada kekurangan. Tahapan-tahapan dari pengujian adalah sebagai berikut:

- a. Graph model yang dibentuk dari Linear Temporal Logic maupun dari event log harus sudah menangani masalah invisible task.
- b. Graph model yang dibentuk dari Linear Temporal Logic maupun dari event log harus sama.

## **6. Penyusunan buku Tugas Akhir**

Pada tahap ini dilakukan pendokumentasian dan pelaporan dari seluruh konsep, dasar teori, implementasi, proses yang telah dilakukan, dan hasil-hasil yang telah didapatkan selama pengerjaan Tugas Akhir.

## **1.7 Sistematika Penulisan**

Buku Tugas Akhir ini bertujuan untuk mendapatkan gambaran dari pengerjaan Tugas Akhir ini. Selain itu, diharapkan dapat berguna untuk pembaca yang tertarik untuk melakukan pengembangan lebih lanjut. Secara garis besar, buku Tugas Akhir terdiri atas beberapa bagian seperti berikut ini.

### **Bab I Pendahuluan**

Bab ini berisi latar belakang masalah, tujuan dan manfaat pembuatan Tugas Akhir, permasalahan, batasan masalah, metodologi yang digunakan, dan sistematika penyusunan Tugas Akhir.

### **Bab II Dasar Teori**

Bab ini membahas beberapa teori penunjang yang berhubungan dengan pokok pembahasan dan yang menjadi dasar dari pembuatan Tugas Akhir ini.

### **Bab III Metode Pemecahan Masalah**

Bab ini membahas cara penulis memecahkan masalah yang ada. Penjelasan tentang algoritma yang dikembangkan penulis dan langkah-langkahnya sehingga dapat memecahkan masalah yang ada.

#### **Bab IV Analisis dan Perancangan Sistem**

Bab ini membahas mengenai perancangan perangkat lunak. Perancangan perangkat lunak meliputi perancangan alur, proses dan perancangan antarmuka pada perangkat lunak.

#### **Bab V Implementasi**

Bab ini berisi implementasi dari perancangan perangkat lunak perangkat lunak dan implementasi fitur-fitur penunjang perangkat lunak.

#### **Bab VI Pengujian dan Evaluasi**

Bab ini membahas pengujian dengan metode pengujian subjektif untuk mengetahui penilaian aspek kegunaan (*usability*) dari perangkat lunak dan pengujian fungsionalitas yang dibuat dengan memperhatikan keluaran yang dihasilkan serta evaluasi terhadap fitur-fitur perangkat lunak.

#### **Bab VII Kesimpulan**

Bab ini berisi kesimpulan dari hasil pengujian yang dilakukan. Bab ini membahas saran-saran untuk pengembangan sistem lebih lanjut.

#### **Daftar Pustaka**

Merupakan daftar referensi yang digunakan untuk mengembangkan Tugas Akhir.

#### **Lampiran**

Merupakan lampiran yang digunakan untuk menjabarkan hasil pengujian algoritma.



## **BAB II**

### **DASAR TEORI**

Pada bab ini akan dibahas mengenai teori-teori yang menjadi dasar dari pembuatan Tugas Akhir.

#### **2.1 *Process Discovery***

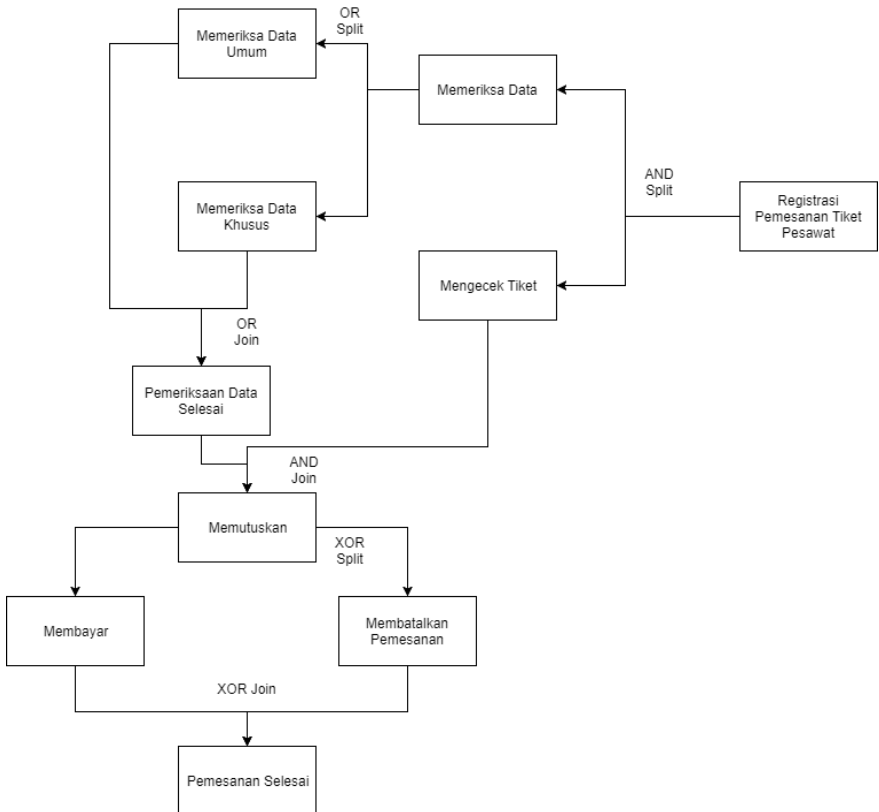
*Proses mining* adalah suatu ilmu yang menggabungkan antara pembelajaran mesin serta *data mining* pada satu sisi dan memodelkan serta menganalisa proses pada sisi yang lain. Ide dasar dari *process mining* adalah menemukan, memantau serta meningkatkan proses sebenarnya dengan mengambil pengetahuan dari *event log* yang ada pada suatu sistem [11].

Proses mining dibagi menjadi tiga bagian, salah satunya adalah *process discovery*. *Process discovery* adalah suatu teknik yang memanfaatkan *event log* dalam penentuan proses model tanpa menggunakan informasi-informasi lainnya. *Process discovery* yang ada diantaranya algoritma *Alpha#* [2] serta *Heuristics Miner* [3].

Proses model yang benar adalah proses model yang mampu menampilkan konkurensi serta *control-flow pattern* [1]. Konkurensi berarti aktivitas pada proses model tidak ditampilkan secara redundan (tidak ada aktivitas yang sama ditampilkan pada proses model) [11]. *Control-flow pattern* adalah penanda relasi antar aktivitas. Terdapat empat relasi antar aktivitas yaitu relasi *sequence*, relasi XOR, relasi AND and relasi OR [11].

Relasi *sequence* adalah relasi yang terjadi untuk menghubungkan satu aktivitas dengan satu aktivitas lainnya. Relasi XOR adalah relasi yang terjadi yang hanya memperbolehkan salah satu aktivitas yang terjadi pada suatu proses. Relasi OR adalah relasi yang terjadi apabila beberapa aktivitas yang terjadi pada suatu proses. Sedangkan relasi AND adalah relasi yang terjadi dimana semua aktivitas harus dijalankan semuanya. Apabila aktivitas pilihan pada relasi XOR, OR dan AND adalah aktivitas sebelum dari aktivitas lainnya, maka *control-flow pattern* dari relasi tersebut adalah penambahan kata “Split” pada relasi [11]. Sedangkan, apabila

aktivitas pilihan pada relasi XOR, OR dan AND adalah aktivitas sebelum dari aktivitas lainnya, maka *control-flow pattern* dari relasi tersebut adalah penambahan kata “Join” pada relasi [11]. Gambar 2.1 merupakan contoh penggunaan relasi pada proses model.



**Gambar 2.1 Contoh Model Proses**

## 2.2 Streaming Event Log

*Event log* merupakan catatan kejadian dari suatu proses bisnis yang berjalan [11]. Sebuah kejadian menunjukkan aktivitas ataupun

langkah dari suatu proses. *Event log* menunjukkan proses tunggal sehingga sebuah kejadian termasuk dalam sebuah proses, yang biasa disebut dengan *case*[1]. *Event log* dapat menyimpan informasi yang berhubungan dengan kejadian, seperti *timestamp* atau *resources*[1]. *Timestamp* adalah waktu terjadinya suatu kejadian. Sedangkan *resources* adalah pelaku yang mengeksekusi suatu kejadian. Definisi lengkap dari *event log* dapat dilihat pada Gambar 2.2[1].

### **Definisi *Event log***

*Event log* terdiri dari rangkaian aktivitas dan waktu yang didefinisikan dengan  $L_{A,TD} = (E, C, \alpha, \gamma, \beta, >)$  dimana:

- E adalah kumpulan kejadian.
- C adalah kumpulan case.
- $\alpha: E \rightarrow A$  adalah fungsi yang menghubungkan setiap kejadian dengan sebuah aktifitas.
- $\gamma: E \rightarrow TD$  adalah fungsi yang menghubungkan setiap kejadian dengan sebuah waktu kejadian (*timestamp*).
- $\beta: E \rightarrow C$  adalah fungsi yang menghubungkan sebuah kejadian dengan sebuah case.
- $> \subseteq E \times E$  adalah *relation succesion*, yang merupakan total suatu kejadian yang dilanjutkan dalam kejadian lain dan termasuk dalam E.  $e_2 > e_1$  adalah notasi singkat untuk  $(e_2, e_1) \in >$ . Kumpulan kejadian yang berkaitan dalam sebuah case disebut sebagai *trace*.

Gambar 2.2 Definisi *Event Log*

Sedangkan *streaming event log* merupakan *event log* yang diobservasi satu per satu berdasarkan waktu dilaksanakannya kejadian pada *event log* tersebut [6]. Sehingga, *streaming event log* adalah *event log* yang tercatat secara *real-time*.

## **2.3 Declarative Miner**

Penemuan *Declarative Miner* secara umum dan model Declare secara khusus dapat digunakan untuk secara efektif

menyelesaikan dua kelemahan penting dari teknik penemuan proses yang ada:

- Model yang dihasilkan cenderung besar dan kompleks terutama di lingkungan yang fleksibel dimana eksekusi proses melibatkan banyak alternatif
- Menawarkan kemungkinan terbatas untuk memandu proses penambangan menuju sifat-sifat menarik yang spesifik

Dengan menggunakan *Declarative miner*, perilaku proses digambarkan sebagai seperangkat aturan yang harus dipenuhi selama proses eksekusi. Penemuan *Declarative miner* dapat dengan mudah dipandu dalam kerangka aturan. Contoh *Declarative miner* dapat dilihat di Tabel 2.1

**Tabel 2.1Potongan *Template Declare***

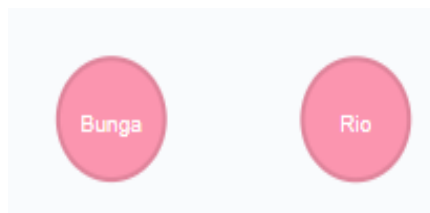
<b><i>Template Declare</i></b>	<b>LTL</b>	<b>Deskripsi</b>
Init(K)	K	K adalah aktivitas pertama yang terekam dalam proses.
Chain Response(K,R)	$\square ( K \rightarrow O ( R ) )$	Jika K terekam, maka aktivitas yang langsung terekam selanjutnya adalah R
Exclusive Choice (K,R)	$( \diamond ( K \vee R ) \wedge ! ( \diamond ( K \wedge R ) ) )$	K atau R akan terekam dalam proses, akan tetapi kedua aktivitas tersebut tidak dapat terekam pada proses yang sama.
Response(K,R)	$\square ( K \rightarrow \diamond ( R ) )$	Aktivitas R hanya boleh terekam apabila aktivitas K sudah terekam dalam proses

<b>Template Declare</b>	<b>LTL</b>	<b>Deskripsi</b>
Existences2(K)	$\diamond ( K \wedge O ( \diamond (K) ) )$	Aktivitas K muncul sekurang-kurangnya dua kali di proses

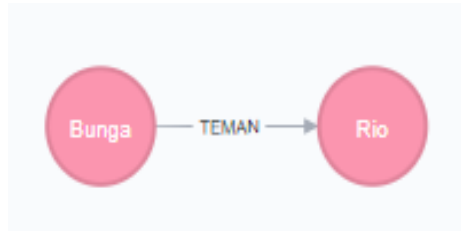
## 2.4 NEO4J

Neo4j adalah aplikasi gratis NoSQL graph database yang di implementasikan di java dan scala dengan pembuatan mulai 2003, dan telah tersedia untuk umum sejak 2007. Neo4j sekarang digunakan oleh ratusan ribu perusahaan dan organisasi di hampir semua industri. Use case yang meliputi matchmaking, manajemen jaringan, analisis perangkat lunak, penelitian ilmiah, routing, manajemen organisasi dan proyek, rekomendasi, jejaring sosial dan lainnya. Neo4j menerapkan model grafik properti secara efisien dalam tingkat penyimpanan.

Neo4j dapat menyimpan 2 hal yang terdiri dari: *Node Label* dan *Relationship Types*. Node merupakan kumpulan data yang berisi informasi-informasi yang tersimpan dalam sebuah data. Contoh: Sebuah database Person yang menampung informasi seperti Nama dan Asal. Sedangkan Relationship merupakan kumpulan data yang berisi hubungan / relasi antar node. Contoh node dan relasi dapat dilihat di Gambar 2.3 dan Gambar 2.4.



**Gambar 2.3. Contoh Node dalam Neo4J**

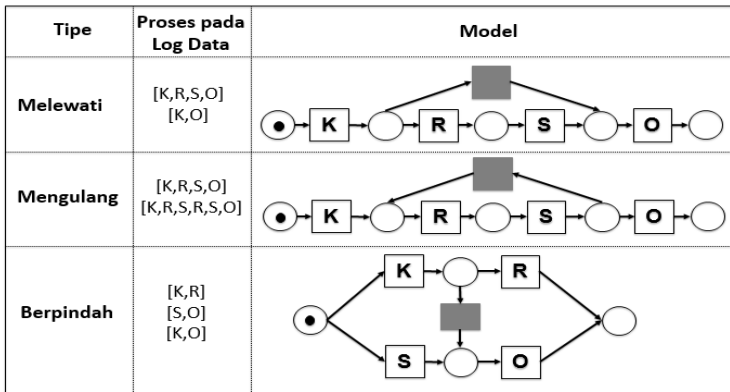


**Gambar 2.4. Contoh Relationship dalam Neo4J**

## **2.5 Invisible Task**

Invisible Task (Aktivitas Tak Terlihat) adalah aktivitas tambahan yang tidak muncul di log data tetapi ditampilkan dalam model [1]. Kegunaan dari invisible task adalah untuk membantu menggambarkan proses secara sebenarnya dalam model proses. Invisible task dibagi menjadi dua macam, yaitu invisible prime task dan invisible non-prime task. Invisible prime task adalah invisible task yang ditangani oleh algoritma Alpha#[1]. Terdapat tiga macam invisible task prime yaitu Melewati, Mengulang, dan Berpindah.

Invisible task tipe Melewati digunakan untuk menggambarkan aktivitas yang dapat dilewati. Invisible task tipe Mengulang digunakan untuk menggambarkan aktivitas yang dapat diulang. Invisible task tipe Berpindah digunakan untuk perpindahan eksekusi pada beberapa percabangan. Contoh dari Invisible task dalam bentuk Petri Net digambarkan pada Gambar 2.5. Kotak yang berwarna abu-abu merupakan contoh *Invisible task*.







Gambar 2.5 Contoh Invisible Task

## 2.6 Linear Temporal Logic (LTL)

Linear Temporal Logic (LTL) [1] adalah bahasa yang menggambarkan logika temporal yang mengacu pada waktu. Linear Temporal Logic dibangun dari konstanta (benar dan salah), sekelompok proposisi atomic, operator logika ( $\neg$ ,  $\vee$ ,  $\wedge$ ,  $\rightarrow$ ) dan operator modal temporal. Terdapat empat operator modal temporal yang digunakan pada Linear Temporal Logic. Penjelasan mengenai operator tersebut dapat dilihat pada Gambar 2.6.

Pada penggalan proses, LTL dibentuk menjadi relasi aktivitas yang disebut *template Declare* [1]. Contoh *template declare* dapat dilihat pada Tabel 2.1. *Template* ini yang digunakan oleh algoritma *Declarative Miner* [1] untuk membentuk model menggunakan log data. Model tersebut berupa graph yang memiliki relasi seperti pada *template Declare*.

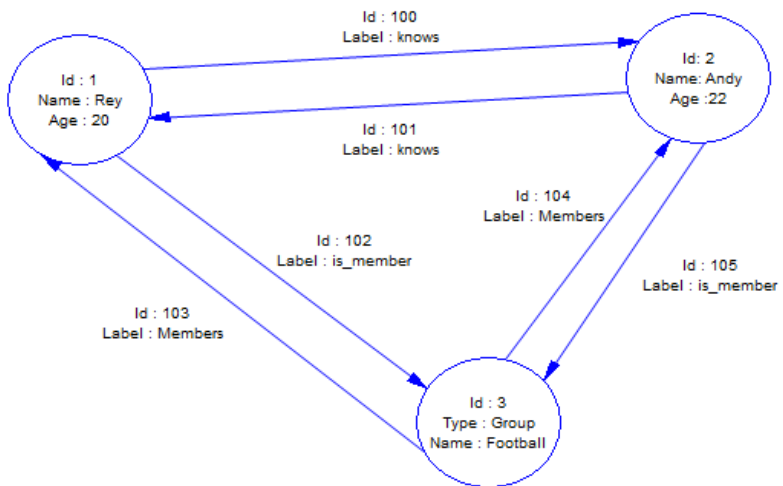
<u>Tekstual</u>	<u>Simbol</u>	<u>Penjelasan</u>	<u>Diagram</u>
$X\gamma$	$\bigcirc\gamma$	$\gamma$ harus dilaksanakan sebagai state selanjutnya	
$G\gamma$	$\square\gamma$	$\gamma$ harus berada pada seluruh state berikutnya	
$F\gamma$	$\diamond\gamma$	$\gamma$ harus berada pada tempat lain di dalam path.	
$\gamma U \phi$	$\gamma U \phi$	$\gamma$ harus dilaksanakan sampai $\phi$ muncul.	

**Gambar 2.6. Operator Modal Temporal (LTL)**

## 2.7 Graph Database

Graph database adalah relasi driven. Melihat struktur data, itu akan menjadi grafik terarah dalam arti matematis. Dibandingkan dengan SQL, mereka sedikit berbeda. Dengan membandingkan deretan data SQL dan entitas dari database grafik, dapat dilihat bahwa database grafik memiliki perbedaan yang berbeda. Data akan terdiri dari node dan simpul, seperti yang ditunjukkan pada Gambar 2.7. Simpul akan memiliki semua informasi tentang beberapa objek. Dan simpul akan mewakili hubungan antar objek. Vertikal akan memiliki nama kata kerja. Misalnya, id subjek: "1" mengetahui sebuah objek dengan id: "2". Inti dari model data ini adalah untuk mengatasi masalah memiliki terlalu banyak hubungan dalam data tabel. Orang akan berakhir menulis query SQL yang sangat kompleks untuk mendapatkan data dengan banyak tingkat hubungan. Juga menganalisa data seperti grafik sangat alami bagi orang untuk melihat pola relasi, yang akan sulit diperhatikan dalam data tabular





**Gambar 2.7. Stuktur Data Graph Database**

*[Halaman ini sengaja dikosongkan]*

## **BAB III**

### **METODE PEMECAHAN MASALAH**

Pada bab ini akan dibahas mengenai metodologi pemecahan masalah yang digunakan sebagai dasar solusi dari pembuatan Tugas Akhir. Metodologi tersebut menerangkan langkah demi langkah proses hingga dapat menghasilkan proses model dari suatu *event log*.

#### **3.1 Cakupan Permasalahan**

Permasalahan utama yang diangkat pada pembuatan Tugas Akhir ini adalah menemukan proses model yang tepat (*process discovery*) dari *streaming event log* yang memiliki *invisible task* yang merupakan sebuah anomali. Baik anomali tersebut adalah melewati (*skip decision*), mengulang (*redo*), dan berpindah (*switch*). Anomali-anomali tersebut sudah dijelaskan di Bab 2.5.

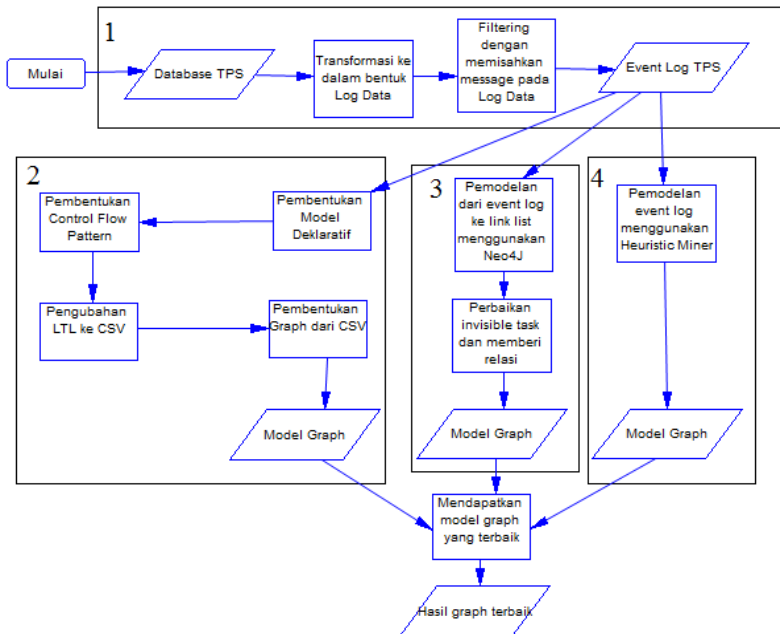
Permasalahan pertama yang dipecahkan dalam Tugas Akhir ini adalah menggambarkan model proses beserta relasinya yang dihasilkan dari *Declarative Miner*.

Permasalahan kedua yang dipecahkan pada Tugas Akhir ini adalah memodelkan proses beserta relasinya dengan input event log yang merupakan sebuah file dalam bentuk *comma delimiter (csv)*.

Permasalahan ketiga yang dipecahkan pada Tugas Akhir ini adalah menggambarkan model proses beserta relasinya yang dihasilkan dari *Heuristic Miner*. Setelah menggambarkan ketiga graph model tersebut, maka kita akan tahu metode mana yang tepat untuk menggambarkan sebuah *business process model*.

Gambar 3.1 adalah gambaran umum alur proses pengerjaan Tugas Akhir. Kotak yang bernomor 1 adalah tahapan *pre-processing* dimana terdapat tahapan untuk mentransformasi dari database ke event log dan memisahkan *message* dari event log sehingga dihasilkan event log tanpa *message*. Kemudian setelah melakukan tahapan *pre-processing* kita melakukan kotak yang bernomor 2,3, dan 4. Kotak yang

bernomor 2 ialah bagian dimana kita akan membangun *control-flow pattern* dari model deklaratif kemudian membentuk *graph* di *Neo4J*. Kotak yang bernomor 3 ialah dimana kita akan membentuk *graph model* langsung dari *event log* menggunakan *Neo4J*, sedangkan kotak yang bernomor 4 ialah dimana kita akan membentuk *graph model* menggunakan *heuristic miner*. Setelah semua itu dilakukan, maka kita akan bisa membandingkan dan mendapatkan *graph* mana yang terbaik dalam memodelkan *event log*.



**Gambar 3.1. Alur Proses Pengerjaan Tugas Akhir**

## 3.2 Masukan

### 3.2.1 Data

Dalam penelitian ini, log data yang digunakan adalah log data mengenai proses impor barang di Terminal Peti Kemas Surabaya. Log data yang didapatkan merupakan hasil transformasi database proses impor barang mulai dari proses yang berjalan dari bulan Januari sampai proses yang berakhir di bulan Maret 2016. Log data yang digunakan mengandung ID\_Case (nomor proses), nama aktivitas dan nama *message*, originator (pelaku aktivitas), sender receiver (pelaku *message*), waktu pelaksanaan, lampiran, detail lampiran, serta biaya pelaksanaan aktivitas.

### 3.3 Pre-processing

#### 3.3.1 Transformasi ke Log Data berdasarkan Database

Log data yang akan digunakan adalah hasil dari transformasi dari database TOS. Log data yang digunakan berisi Case\_ID, nama aktivitas atau *message*, waktu pelaksanaan, pelaku aktivitas (*originator*), pelaku *message* (sender receiver), *cost*, *atribut* tambahan yang mendukung aktivitas (*input*, *output*, *detail attachment*). Cara transformasi secara garis besar dapat dilihat pada Tabel 3.1.

**Tabel 3.1. Cara Transformasi ke Log Data secara Garis Besar**

Isi Log Data	Cara Transformasi
Case_ID	Mengambil data pada kolom CONTAINER_KEY
Waktu pelaksanaan	Melakukan <i>inverse</i> distribusi kumulatif normal berdasarkan database dan rentang waktu intervensi dari <i>expert</i>
Nama aktivitas dan <i>message</i>	Berdasarkan intervensi <i>expert</i>
Pelaku <i>message</i> (sender	

receiver)	
<i>Cost</i>	Melakukan <i>inverse</i> distribusi kumulatif normal berdasarkan list harga pengeluaran proses secara keseluruhan
Atribut tambahan ( <i>input output</i> )	Berdasarkan list dokumen yang dibutuhkan pada proses impor barang
Atribut tambahan ( <i>detail attachment</i> )	Berdasarkan kolom pada database, yaitu CTR_TYPE, HAS_QUARANTINE_FLAG, dan CUSTOM_BEHANDLE_COUNT

#### a. Transformasi ke Log Data untuk Pengisian Waktu

##### Pelaksanaan

Database TOS menyimpan waktu dari aktivitas dan *message*, akan tetapi tidak semua kolom menyimpan waktu setiap aktivitas dan *message*. Terdapat beberapa kolom yang menunjukkan rentang waktu pelaksanaan beberapa aktivitas serta *message* sekaligus. Sedangkan, log data membutuhkan waktu setiap aktivitas atau *message*. Oleh karena itu, diperlukan perkiraan waktu untuk setiap aktivitas dan *message*.

Selain database TOS, *expert* juga memberikan perkiraan rentang waktu eksekusi setiap aktivitas. Perkiraan rentang waktu eksekusitersebut dimanfaatkan untuk menentukan perkiraan waktu setiap aktivitas dan *message*. Penentuan waktu eksekusi setiap aktivitas menggunakan *inverse normal distribution* yang mengacu pada perkiraan rentang waktu eksekusi dan rentang waktu beberapa aktivitas dari database. Penggunaan *normal distribution* dipilih dengan asumsi bahwa persebaran datanya adalah sebaran normal.

Persamaan untuk perhitungan waktu aktivitas dapat dilihat pada Persamaan (1) sampai Persamaan (3). Waktu aktivitas didapatkan dari penambahan waktu sebelumnya dan perkiraan

waktu eksekusi aktivtias menggunakan *inverse normal distribution*. Rata-rata untuk *inverse normal distribution* adalah nilai perkiraan berdasarkan median rentang waktu eksekusi dan rentang waktu beberapa aktivitas database. Simpangan baku adalah 5% dari nilai rata-rata. Pemilihan 5% untuk memberikan persebaran yang tidak lebar dalam hasil perkiraan waktu eksekusi aktivitas.

$$\begin{aligned} time(Act\_now) &= time(Act\_before) \\ &+ NormInv(p, \mu, \sigma_x) \end{aligned} \quad 1)$$

$$\mu = \frac{\widetilde{Es. \overline{Act}}}{\sum_{i=0}^n \widetilde{Es. \overline{Act}_i}} \times \overline{ActinLog} \quad 2)$$

$$\sigma_x = 0,05 \times \mu \quad 3)$$

dimana:

$time(Act\_now)$	= waktu pelaksanaan aktivitas atau message sekarang
$time(Act\_bfeore)$	= waktu pelaksanaan aktivitas atau message sebelumnya
$NormInv$	= inverse normal distribution untuk memperkirakan rentang waktu tiap aktivtias
$\widetilde{Es. \overline{Act}}$	= median dari interval rentang waktu yang didapatkan dari expert
$\overline{ActinLog}$	= rata-rata dari rentang waktu kelompok aktivitas di database
$n$	= jumlah aktivitas yang berada pada log data
$p$	= nilai probabilitas secara random
$\mu$	= rata-rata untuk perhitungan inverse normal distribution

$$\sigma_x = \text{simpangan baku untuk perhitungan inverse normal distribution}$$

Contoh dari pembentukan waktu pelaksanaan adalah pembentukan waktu aktivitas-aktivitas yang terjadi saat truk datang untuk mengambil barang (TRUCK IN) dan truk keluar dari Terminal Peti Kemas (TRUCK OUT). Perhitungan dapat dilihat pada Tabel 3.2. Batas atas dan batas bawah adalah intervensi yang diberikan oleh *expert* mengenai rentang waktu eksekusi setiap aktivitas. Kemudian, dari rentang waktu eksekusi tersebut, diambil nilai median yang ditunjukkan oleh nilai pada Median(Es.Act). Kemudian, nilai rata-rata didapatkan dari nilai pada Median(Es.Act) dibagi jumlah keseluruhan nilai media dan dikali dengan rentang waktu kelompok aktivitas pada database, yang ditunjukkan oleh nilai ActInLog. Kemudian, nilai standar deviasi menunjukkan nilai simpangan baku. Nilai rata-rata dan simpangan baku yang didapatkan kemudian diolah menggunakan Persamaan (7) untuk menghasilkan waktu setiap aktivitas atau *message*. Hasil waktu setiap aktivitas dapat dilihat pada Tabel 3.3.

**Tabel 3.2. Perhitungan Waktu Aktivitas untuk Log Data**

Aktivitas	Batas Atas	Batas Bawah	Median(Es.Act)	Rata-rata	Standar Deviasi
Dispatch WQ Delivery to CHE	0:00:30	0:00:50	0:00:40	0:00:40	0:00:02
Determine Container Type	0:01:00	0:02:00	0:01:30	0:01:31	0:00:05
Determining Uncoitaner	0:01:00	0:02:00	0:01:30	0:01:31	0:00:05
Decide Task Before Lift Container	0:01:00	0:02:00	0:01:30	0:01:31	0:00:05
Prepare Tools	0:10:00	0:15:00	0:12:30	0:12:35	0:00:38
Lift on Container Truck	0:02:00	0:03:00	0:02:30	0:02:31	0:00:08
Truck Go To Gate Out	0:01:00	0:02:00	0:01:30	0:01:31	0:00:05
Check Container before Truck out	0:01:00	0:02:00	0:01:30	0:01:31	0:00:05
		sum(median(Es.Act))	0:23:10		
		ActInLog	0:23:20		



**Tabel 3.3. Contoh Hasil Estimasi Waktu Setiap Aktivitas**

CaseID	Activity	Time	NormInv
4619882	Truck in	3/14/16 19:30:21	
4619882	Dispatch WQ Delivery to CHE	3/14/16 19:31:02	0:00:41
4619882	Determine Container Type	3/14/16 19:32:30	0:01:28
4619882	Determining Uncontainer	3/14/16 19:34:06	0:01:36
4619882	Decide Task Before Lift Container	3/14/16 19:35:36	0:01:31
4619882	Prepare Tools	3/14/16 19:48:30	0:12:54
4619882	Lift on Container Truck	3/14/16 19:51:05	0:02:35
4619882	Truck Go To Gate Out	3/14/16 19:52:34	0:01:29
4619882	Check Container before Truck out	3/14/16 19:54:13	0:01:39
4619882	Truck Out	3/14/16 22:48:25	

### b. Transformasi ke Log Data untuk Pengisian *Detail*

#### *Attachment*

*Detail attachment* pada log data berisi atribut atau data tambahan berkaitan dengan suatu aktivitas atau *message*. *Detail attachment* akan digunakan dalam penentuan aktivitas pengganti dari aktivitas hilang pada metode perbaikan proses yang terpotong. *Detail attachment* diambil dari data pada database TOS, yaitu data pada kolom CTR\_TYPE, HAS\_QUARANTINE\_FLAG, CUSTOM\_BEHANDLE\_COUNT. Pseudocode dari pengisian *detail attachment* ditunjukkan pada Gambar 3.2. Setiap data di database TOS akan dicek dan *detail attachment* akan terisi sesuai dengan aturan pada pseudocode.

```

for all data in database TOS:
    if CTR_TYPE is "DRY":
        DetailAttachmentInLog.add("Dry")
    else if CTR_TYPE is "RFR":
        DetailAttachmentInLog.add("Reefer")
    else:
        DetailAttachmentInLog.add("Uncontainer")
    if CUSTOM_BEHANDLE_COUNT is "0":
        DetailAttachmentInLog.add("Green Line")
    else:
        DetailAttachmentInLog.add("Red Line")

```

```

if HAS_QUARANTINE_FLAG is "YES":
    DetailAttachmentInLog.add("Quarantine")

```

**Gambar 3.2. Pseudocode Pengisian Detail Attachment**

### 3.3.2 Filtering List Aktivitas pada Log Data

Tahapan kedua dari *pre-processing* adalah *filtering* list aktivitas pada log data. *Filtering* dilakukan dengan cara memisahkan data *message* dari log data, sehingga log data hanya berisi data aktivitas. Hal ini dilakukan karena algoritma pemodelan proses mengolah aktivitas, bukan *message*.

Pemisahan *message* pada log data didasarkan pada data pelaksana. Apabila pelaksana ada dua yaitu *sender* dan *receiver*, maka data itu adalah data *message*. Sedangkan apabila pelaku hanya satu, yang biasa disebut *originator*, maka data tersebut adalah data aktivitas. Pseudocode untuk penentuan *message* dan aktivitas dapat dilihat pada Gambar 3.3. Apabila data terdeteksi *message*, maka akan ditambahkan pada *log\_message* dan dihapus dari *log\_data*. Penambahan *message* tidak hanya melibatkan nama *message*, melainkan waktu pelaksanaan serta atribut yang berkaitan dengan *message* tersebut.

```

activity_or_message = data_in_column_activity_of_Log
for x in activity_or_message:
    if sender(x) and receiver(x) != NULL:
        Log_message.add(message)
        Log_data.erase(message)

```

**Gambar 3.3. Pseudocode pemisahan aktivitas dan message**

### 3.3.3 Pemisahan Log Data

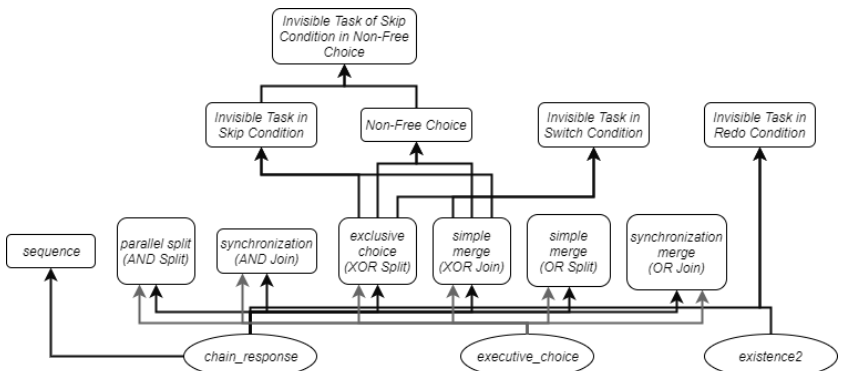
Proses yang terpotong dapat terjadi dikarenakan pengambilan log data dalam interval waktu tertentu. Log data yang ditransformasi dari database TOS berisi proses-proses lengkap yang berjalan di bulan Januari sampai bulan Maret. Untuk mendapatkan log data yang berisi proses yang terpotong, maka log data dipecah menjadi tiga log data, yaitu log data yang berisi aktivitas yang berjalan pada bulan Januari, log data yang

berisi aktivitas yang berjalan pada bulan Februari, dan log data yang berisi aktivitas yang berjalan pada bulan Maret.

### 3.4 Pembentukan *Business Process Model*

#### 3.4.1 Pembentukan *Business Process Model* dari Model Deklaratif

Pada penggambaran *control-flow patterns*, model deklaratif yang digunakan menggunakan *rules* atau *template* berupa *chain\_response*, *executive\_choice*, *response*, dan *existence(2)*. Tidak semua *control-flow patterns* ditentukan hanya berdasarkan *rules* pada model deklaratif, melainkan juga berdasarkan *control-flow patterns* lain yang sudah terbentuk. Gambar 3.4 menunjukkan alur pembentukan *control-flow patterns*.



**Gambar 3.4. Alur Hirarki Pembentukan Control-Flow Patterns**

Berdasarkan Gambar 3.4, metode pembentukan *control-flow patterns* dapat dilihat pada Tabel 3.4. Di dalam metode, terdapat beberapa variabel tambahan. Variabel-variabel tersebut adalah *total\_after(act)*, *total\_before(act)*, *total\_ex\_after(act)*, *total\_ex\_before(act)*, *total\_chain\_after(act)*, dan *total\_chain\_before(act)*.

Total\_next(act) adalah jumlah *chain\_response*(act, aktivitas lainnya) yang memiliki nilai *support* lebih dari 0. Sedangkan total\_before(act) adalah jumlah *chain\_response*(aktivitas lainnya, act) yang memiliki nilai *support* lebih dari 0. Total\_chain\_before(act) adalah jumlah *chain\_response*(aktivitas sebelum act, aktivitas sebelum act) yang memiliki nilai *support* lebih dari 0. Total\_chain\_after(act) adalah jumlah *chain\_response*(aktivitas setelah act, aktivitas setelah act) yang memiliki nilai *support* lebih dari 0. Total\_ex\_after(act) adalah jumlah *executive\_choice* (aktivitas setelah act, aktivitas setelah act) yang memiliki nilai *support* lebih dari 0 dan total\_chain\_before(act) adalah jumlah *executive\_choice*(aktivitas sebelum act, aktivitas sebelum act) yang memiliki nilai *support* lebih dari 0.

Sebagai contoh, *chain\_response*(K,R), *chain\_response*(K,S), dan *chain\_response*(T,S) tergambar dalam model deklaratif. Dari *rules* pada model deklaratif tersebut, total\_next(K) dan total\_before(S) adalah dua, dan total\_next(T) dan total\_before(R) adalah satu, serta total\_ex\_after(K) adalah satu.

**Tabel 3.4. Metode untuk membangun control-flow-patterns**

Pattern	Peraturan untuk membangun <i>control-flow patterns</i>
Exclusive choice (XOR Split)	If total_next(act)>1 and total_ex_after(act)>0 and(total_chain_after(act) < total_after(act)): act -> O ( ( y1 ∨ y2 .... ∨ yn ) )
Simple Merge (XOR join)	If total_before(act)>1 and total_ex_before(act)>0 and(total_chain_before(act)< total_next(act)): O ( ( y1 ∨ y2 .... ∨ yn ) ) -> O (

Pattern	Peraturan untuk membangun <i>control-flow patterns</i>
	act )
Invisible Task in Skip and Switch Condition	<p>If act appears in Exclusive Choice before “-&gt;” and appears in Simple Merge before “-&gt;”:  Change <math>O((act \vee y_2 \dots \vee y_n)) \rightarrow O(other\_act)</math> into <math>O((Invisible\_Task \vee y_2 \dots \vee y_n)) \rightarrow O(other\_act)</math></p> <p>If act appears in Exclusive Choice after “-&gt;” and appears in Simple Merge after “-&gt;”:  Change <math>other\_act \rightarrow O((act \vee y_2 \dots \vee y_n))</math> into <math>other\_act \rightarrow O((Invisible\_Task \vee y_2 \dots \vee y_n))</math></p>
Invisible Task in Redo Condition	<p>If act appears in LTLSequence before “-&gt;” and</p> <p style="padding-left: 40px;">act not in <i>Existence</i>(2) and  act_after(act) in <i>Existence</i>(2):</p> <p style="padding-left: 80px;">act <math>\rightarrow \Diamond((act\_after1 \wedge O(act\_after2) \dots \wedge O(act\_after_n)))</math></p>

Setelah mendapatkan LTL kemudian kita ubah ke dalam bentuk cypher untuk mendapatkan sebuah graph model. Kita mencari relasi antar aktivitas juga disini dengan melalui jumlah incoming relation dan outgoing relation dari aktivitas. Jumlah Incoming

relation adalah Jumlah aktivitas yang terjadi sebelumnya, sebaliknya dengan outgoing relation adalah jumlah aktivitas yang sebelumnya. Pada tabel 3.5 ditunjukkan bagaimana cara mengubah LTL ke dalam Cypher.

**Tabel 3.5. Mengubah LTL ke Neo4J**

<b>LTL</b>	<b>Cypher</b>
act -> O(y)	<i>for i=0 until i=count_activity create relation activity[i]- [r:NEXT]-&gt;activity[i+1]</i>
act -> O ( ( y1 $\vee$ y2 .... Vyn ) )	<i>match(activity[i])- [r:NEXT]-&gt;(activity[i+1])  if outgoing(activity[i])&gt;1 and ingoing(activity[i+1])=1  create relation activity[i]- [r:XORSPLIT]-&gt; activity [i+1]</i>
O ( ( y1 $\vee$ y2 .... Vyn ) ) - > O ( act )	<i>match(activity[i])- [r:NEXT]-&gt;(activity[i+1])  if outgoing(activity[i])=1 and ingoing(activity[i+1])&gt;1  create relation activity[i]- [r:XORJOIN]-&gt; activity [i+1]</i>
Skip Decision and Switch: act appears in Exclusive Choice before “->” and appears in Simple	<i>Skip Condition: match(activity[i])- [r:NEXT]-&gt;(activity[i+1])  where</i>

LTL	Cypher
<p>Merge before “-&gt;”</p> <p>Change <math>O((\text{actV } y2 \dots \text{Vyn})) \rightarrow O(\text{other\_act})</math> into</p> <p style="padding-left: 40px;"><math>O((\text{Invisible\_TaskV } y2 \dots \text{Vyn})) \rightarrow O(\text{other\_act})</math></p>	<p><i>outgoing(activity[i])&gt;1 and ingoing(activity[i+1])&gt;1 and outgoing(activity[i+1])&gt;1</i></p> <p style="padding-left: 40px;"><i>create node i:invisible task create relation activity[i]-(Next)-&gt;I Create relation i-(Next)-&gt;activity[i+1] Delete relation activity[i]-(Next)-&gt;activity[i+1]</i></p>
<p>act appears in Exclusive Choiceafter “-&gt;” and appears in Simple Merge after “-&gt;”</p> <p>LTL: Change other_act -&gt; <math>O((\text{actV } y2 \dots \text{Vyn}))</math> into</p> <p style="padding-left: 40px;">other_act -&gt; <math>O((\text{Invisible\_TaskV } y2 \dots \text{Vyn}))</math></p>	<p><i>Switch Condition:</i></p> <p style="padding-left: 40px;"><i>match(activity[i])-[r:NEXT]-&gt;(activity[i+1]) where outgoing(activity[i])&gt;1 and ingoing(activity[i+1])&gt;1 create node i:invisible task create relation activity[i]-(Next)-&gt;I Create relation i-(Next)-&gt;activity[i+1] Delete relation activity[i]-(Next)-&gt;activity[i+1]</i></p>
<p>act -&gt; <math>\diamond((\text{act\_after1} \wedge O(\text{act\_after2}) \dots \wedge O(\text{act\_after\_n}))</math></p>	<p><i>match(activity[i])-[r:NEXT]-&gt;(activity[i+1])</i></p> <p style="padding-left: 40px;"><i>create relation activity[i]-[r:NEXT]-&gt;activity[i+1]</i></p>

3.4.2 Pembentukan *Business Process Model* dari Model Neo4J

Untuk mendapatkan sebuah model dari Neo4J, hal pertama yang dilakukan yaitu menginputkan event log yang berekstensi csv ke dalam Neo4J dan dibuat ke dalam link list. Cara untuk menginputkan event log yang berekstensi csv ke dalam Neo4J dan dibuat ke dalam link list terdapat di Gambar 3.5

```
//LOAD CSV
LOAD CSV with headers FROM "file e" AS line
Merge (:Activity
{CaseId:line.Case_ID,Name:line.Activity,StartTime:line.Start_Stamp,EndTime:line.End_Stamp })
LOAD CSV with headers FROM "file:///eventlogswitch.csv" AS line
Merge (:CaseActivity {Name:line.Activity })
//CONNECT THE ACTIVITY
Match (c:Activity)
WITH COLLECT(c) AS Caselist
UNWIND RANGE(0,Size(Caselist) - 2) as idx
WITH Caselist[idx] AS s1, Caselist[idx+1] AS s2
match (b:CaseActivity),(a:CaseActivity)
WHERE s1.CaseId = s2.CaseId AND s1.Name = a.Name AND s2.Name = b.Name
MERGE (a)-[r:NEXT]->(b)
```

Gambar 3.5. Memasukkan event log ke Neo4J

Setelah terbentuk ke dalam link list, maka kita mencari relasi antar aktivitas, mengecek event log apakah mengandung invisible task atau tidak. Jika ada maka kita memperbaiki graph model. Cara untuk mencari relasi ialah dengan jumlah incoming relation dan outgoing relation dari aktivitas. Jumlah Incoming relation adalah Jumlah aktivitas yang terjadi sebelumnya, sebaliknya dengan outgoing relation adalah jumlah aktivitas yang sebelumnya. Pada Tabel 3.6 dapat dilihat metode cypher untuk menemukan sebuah pattern.

Tabel 3.6. Penemuan Pola dalam Neo4J

Pattern	Cypher
Sequence	<i>for i=0 until i=count_activity create relation activity[i]-</i>



Pattern	Cypher
	<i>[r:NEXT]-&gt;activity[i+1]</i>
Exclusive choice (XOR Split)	<i>match(activity[i])- [r:NEXT]-&gt;(</i> <i>activity[i+1])</i>  <i>if outgoing(activity[i])&gt;1 and</i> <i>ingoing(activity[i+1])=1</i>  <i>create relation</i> <i>activity[i]-[r:XORSPLIT]-&gt;</i> <i>activity [i+1]</i>
Simple Merge (XOR join)	<i>match(activity[i])- [r:NEXT]-&gt;(</i> <i>activity[i+1])</i>  <i>if outgoing(activity[i])=1 and</i> <i>ingoing(activity[i+1])&gt;1</i>  <i>create relation</i> <i>activity[i]-[r:XORJOIN]-&gt;</i> <i>activity [i+1]</i>
Invisible Task in Skip and Switch Condition	<i>Skip Condition:</i> <i>match(activity[i])- [r:NEXT]-&gt;(</i> <i>activity[i+1])</i>  <i>where outgoing(activity[i])&gt;1</i> <i>and ingoing(activity[i+1])&gt;1 and</i> <i>outgoing(activity[i+1])&gt;1</i>  <i>create node i:invisible task</i> <i>create relation activity[i]-(Next)-</i> <i>&gt;i</i>  <i>Create relation i-(Next)-</i> <i>&gt;activity[i+1]</i>  <i>Delete relation activity[i]-(Next)-</i> <i>&gt;activity[i+1]</i>

Pattern	Cypher
	<i>Switch Condition:</i> <i>match(activity[i])-&gt;[r:NEXT]-&gt;(</i> <i>activity[i+1])</i> <i>where outgoing(activity[i])&gt;1</i> <i>and ingoing(activity[i+1])&gt;1</i> <i>create node i:invisible task</i> <i>create relation activity[i]-(Next)-</i> <i>&gt;i</i> <i>Create relation i-(Next)-</i> <i>&gt;activity[i+1]</i> <i>Delete relation activity[i]-(Next)-</i> <i>&gt;activity[i+1]</i>
Invisible Task in Redo Condition	<i>match(activity[i])-&gt;[r:NEXT]-&gt;(</i> <i>activity[i+1])</i> <i>create relation activity[i]-</i> <i>[r:NEXT]-&gt;activity[i+1]</i>

### 3.4.3 Pembentukan *Business Process Model* dari Model Heuristic Miner

Kita akan membentuk Business Process Model dari Model Heuristic Miner menggunakan ProM dan Disco. Pertama-tama kita akan menggunakan Disco untuk mengubah event log dari berekstensi csv menjadi mxml dimana ekstensi mxml ini akan digunakan sebagai input di ProM. Di ProM setelah menginputkan event log berekstensi mxml pilih start analyzing kemudian pilih heuristic miner. Setelah itu, kita akan disuruh untuk mengisi nilai dari Relative to best threshold,

Positive observations, Dependency threshold, Length one loops threshold, length two loops threshold, long distance threshold, Dependency divisor, dan And threshold, tapi kita akan memakai nilai default semua. Setelah semua nilai terisi, maka kita melakukan start analyzing.

### 3.5 Keluaran

#### 3.5.1 Metode Pengujian

Pada hasil dari tugask akhir ini terdapat 3 graph yang menggambarkan business process model yaitu graph yang berasal dari LTL, graph yang berasal dari Neo4J, dan graph yang berasal dari heuristic miner. Untuk menguji baik buruk nya metode graph database ini, maka kita mengkomparasi antara 3 graph ini dan mengetahui graph mana yang merupakan paling baik.

#### 3.5.2 Skenario Uji Coba

Skenario uji coba berdasarkan metode pengujian yang dijelaskan pada bab 3.5.1. Adapun detail dari skenario uji coba untuk menguji tugas akhir ini adalah:

1. Menguji apakah *event log* yang terbentuk dari database sudah terbentuk dengan benar.
2. Menguji apakah *event log* yang terbentuk dari *declarative miner* sudah meenangani *invisible task* dengan benar.
3. Menguji apakah *event log* yang terbentuk dari *Neo4J* sudah meenangani *invisible task* dengan benar.
4. Menguji apakah *event log* yang terbentuk dari *heuristic miner* sudah meenangani *invisible task* dengan benar.
5. Membandingkan metode manakah yang terbaik untuk memodelkan *Business Process Model*.

*[Halaman ini sengaja dikosongkan]*

## **BAB IV IMPLEMENTASI**

Bab ini membahas tentang implementasi dari perancangan sistem. Bahasa pemrograman yang digunakan adalah bahasa pemrograman Python.

### **4.1 Lingkungan Implementasi**

Lingkungan implementasi merupakan lingkungan dimana sistem ini dibangun, dimana akan dijelaskan mengenai kebutuhan perangkat yang diperlukan untuk membangun sistem ini. Lingkungan implementasi dibagi menjadi dua, yaitu lingkungan implementasi terhadap perangkat keras dan lingkungan implementasi terhadap perangkat lunak.

#### **4.1.1. Perangkat Keras**

Implementasi dilakukan pada sebuah laptop dengan spesifikasi sebagai berikut:

- Merk : ASUS.
- Seri : N Series 750.
- Prosesor : AMD A8-5545M APU @ 1.70 GHz.
- RAM : 8 GB.

#### **4.1.2. Perangkat Lunak**

Perangkat lunak yang mendukung fungsionalitas sistem adalah:

1. Sistem Operasi Windows  
Sistem operasi yang digunakan adalah Microsoft Windows 8.1 Single Language 64-bit.
2. Java
3. Eclipse
4. ProM
5. Neo4j
6. SQL

## 4.2 Penjelasan Implementasi

Pada sub bab ini dijelaskan implementasi setiap metode yang dijelaskan pada bab metode pemecahan masalah, sehingga terbentuk suatu perangkat lunak yang mengimplementasi metode-metode pada bab metode pemecahan masalah.

### 4.2.1 Implementasi

Sub bab ini membahas implementasi untuk pencarian business process model dimulai dari pre-processing dan penemuan model baik secara *Declarative*, *Neo4J*, maupun secara *Heuristic*.

#### 4.2.1.1 Pre-Processing

Antarmuka Awal Menjalankan Simulasi *Process Discovery* berfungsi sebagai perantara sistem dan pengguna dalam melakukan pemilihan *event log* sebagai data simulasi untuk *process discovery* menggunakan algoritma yang diusulkan. Terdapat beberapa fungsi yang digunakan sebagai perantara tersebut.

##### 4.2.1.1.1 Implementasi Transformasi ke Log Data

Untuk melakukan pre-processing data, langkah pertama yang harus dilakukan ialah transformasi dari database menjadi log data. Disini kita mengimplementasikan metode yang telah dijabarkan di bab 3.3.1. Untuk mengubah data dari database ke event log maka perlu melakukan yang ada di Kode Sumber 4.1. Kode Sumber 4.1 merupakan bagian dari source code bukan semuanya. Disini kita hanya perlu mengganti variabel-variabel yang akan diubah beserta nilainya, dan kalau variabel waktu terdapat perhitungan distribusi terlebih dahulu

```

INSERT INTO `table 4`
(case_id,Sender,Originator,Input,Activity,output,Receiver,actTime,Cost,Yard_block,Yard_slot,De
ail_lampiran)
SELECT CONTAINER_KEY as Case_id,'' as Sender,'customer' as Originator, 'NPWP, SIUP, API, SRP,
TDP, NPIK, IT, INVOICE, PO, SK, BL, COO' as Input,'document_Entry_via_PDE' as Activity, 'BC
2.0' as Output, '' as Receiver,( VESSEL_ATB-((RAND()*(0.125-0.08333333))+0.08333333) ) AS
Time,'0' as Cost, '' as Yard_block, '' as Yard_slot,
(IF(HAS_QUARANTINE_FLAG = 'YES',
(case when CTR_TYPE='DRY' && CUSTOMS_BEHANDLE_COUNT=0 then 'Quarantine;Dry;Green Line'
      when CTR_TYPE='DRY' && CUSTOMS_BEHANDLE_COUNT=1 then 'Quarantine;Dry;Red Line'
      when CTR_TYPE='RFR' && CUSTOMS_BEHANDLE_COUNT=0 then 'Quarantine;Reefer;Green Line'
      when CTR_TYPE='RFR' && CUSTOMS_BEHANDLE_COUNT=1 then 'Quarantine;Reefer;Red Line'
      when CTR_TYPE='OVD' && CUSTOMS_BEHANDLE_COUNT=0 then 'Quarantine;Uncontainer;Green
Line'
      when CTR_TYPE='OVD' && CUSTOMS_BEHANDLE_COUNT=1 then 'Quarantine;Uncontainer;Red
Line'
      ELSE ''
      END
) ,(case when CTR_TYPE='DRY' && CUSTOMS_BEHANDLE_COUNT=0 then 'Dry;Green Line'
      when CTR_TYPE='DRY' && CUSTOMS_BEHANDLE_COUNT=1 then 'Dry;Red Line'
      when CTR_TYPE='RFR' && CUSTOMS_BEHANDLE_COUNT=0 then 'Reefer;Green Line'
      when CTR_TYPE='RFR' && CUSTOMS_BEHANDLE_COUNT=1 then 'Reefer;Red Line'
      when CTR_TYPE='OVD' && CUSTOMS_BEHANDLE_COUNT=0 then 'Uncontainer;Green Line'
      when CTR_TYPE='OVD' && CUSTOMS_BEHANDLE_COUNT=1 then 'Uncontainer;Red Line'
      ELSE ''
      END
) )) as Detail_lampiran

```

### Kode Sumber 4.1. Transformasi Data

#### 4.2.1.1.2 Implementasi Pemisahan Message dari Log Data serta Pengelompokan Data per Bulan

Setelah mendapatkan event log dalam bentuk excel, maka kita perlu memilah data dengan kondisi mana aktivitas mana message. Jika Message maka akan dibuang. Cara untuk implementasi dilakukan melalui fitur filter excel pada Kode Sumber 4.2

Case ID	Sender	Output	Receiver	Time	Cost	Yard Blo	Yard Slot	Detail Lampiran
4691694	Customer	BC 2.0		24/03/2016 20:05	0			Dry;Green Line
4691694	Customer	BC 2.0	SKP	24/03/2016 20:06	0			
4691694				24/03/2016 22:10	48379,74			
4691694				24/03/2016 22:49	690,9082			
4691694				24/03/2016 22:57	74,99			
4691694				24/03/2016 22:59	28,99 R		106	
4691694	SKP	BC 2.0	Customer	26/03/2016 4:16	1,358024			
4691694		BC 2.0		27/03/2016 8:39	8,604293			Dry;Green Line
4691694		SPPB		27/03/2016 22:48	2055,586			
4691694	Pejabat		Customer	27/03/2016 22:50	0			
4691694		CEIR		28/03/2016 14:45	20,41807			
4691694	Customer		TPS	28/03/2016 14:47	0			
4691694				28/03/2016 18:40	8,74889			
4691694				28/03/2016 18:41	0,514016			
4691694				28/03/2016 18:44	1,313631			
4691694				28/03/2016 18:46	23,37			
4691694				28/03/2016 18:48	1,544251			
4691694				28/03/2016 18:50	23,97			

### Kode Sumber 4.2. Filter data dari Message

```

if(declareMinerOutput.getConstraintParametersMap().get(j).get(0).equals(act)
&&declareMinerOutput.getTemplate().get(j).equals(DeclareTemplate.Chain_Response)){
totalafter++;
}

```



### Kode Sumber 4.3. Mencari nilai total\_after

#### a.2. Mencari nilai total\_before

Apabila `declaremineroutput.get(j).get(1)` sama dengan aktivitas dan `declaremineroutput.get(j)` merupakan chain response maka total before bertambah satu.

```
elseif(declareMinerOutput.getConstraintParametersMap().get(j).get(1).equals(act)
&&declareMinerOutput.getTemplate().get(j).equals(DeclareTemplate.Chain_Response)){
                                totalbefore++;
}
```

### Kode Sumber 4.4. Mencari nilai total\_before

#### a.3. Mencari nilai total\_after\_resp

Apabila `declaremineroutput.get(j).get(0)` sama dengan aktivitas dan `declaremineroutput.get(j)` merupakan sebuah response maka total\_after\_resp bertambah sejumlah satu.

```
elseif(declareMinerOutput.getConstraintParametersMap().get(j).get(0).equals(act)
&&declareMinerOutput.getTemplate().get(j).equals(DeclareTemplate.Response)){
                                totalafterresp++;
}
```

### Kode Sumber 4.5. Mencari nilai total\_after\_resp

#### a.4. Mencari nilai total\_before\_resp

Apabila `declaremineroutput.get(j).get(1)` sama dengan aktivitas dan `declaremineroutput.get(j)` merupakan response maka total\_before\_resp bertambah satu.

```
elseif(declareMinerOutput.getConstraintParametersMap().get(j).get(1).equals(act)
```

```

&&declareMinerOutput.getTemplate().get(j).equals(DeclareTemplate.Response)) {
                                totalbefresp++;
}

```

**Kode Sumber 4.6. Mencari nilai total\_before\_resp**

b) Tahap 2

Tahap kedua dijalankan untuk menentukan semua aktivitas yang memiliki relasi Sequence, dan menentukan aktivitas awal dan aktivitas akhir dari suatu proses model. Pertama dilakukan perulangan sebanyak total aktivitas, kemudian terdapat tiga *case*. Kode Sumber 4.7 menunjukkan potongan Declarative Miner pada tahap kedua.

b.1. Mencari relasi Sequence

Apabila ada nilai total\_after yang bernilai satu, maka dibuat koneksi Sequence antara aktivitas tersebut dengan aktivitas sebelumnya.

b.2. Mencari aktivitas akhir

Apabila ada suatu aktivitas yang memiliki nilai total\_after dan total\_after\_resp yang bernilai 0 (nol) maka aktivitas tersebut dikategorikan sebagai aktivitas akhir.

b.3. Mencari aktivitas awal

Apabila ada suatu aktivitas yang memiliki nilai total\_before dan total\_before\_resp yang bernilai 0 (nol) maka aktivitas tersebut dikategorikan sebagai aktivitas awal.

```

if(declareMinerOutput.getConstraintParametersMap().get(j).get(0).equals(act)
&&declareMinerOutput.getTemplate().get(j).equals(
    DeclareTemplate.Chain_Response)) {

    y.add(declareMinerOutput.getConstraintParametersMap().get(j).get(1));
}
}
if(total_after.get(act)==1) {
    LTLtemp.add(act);
}

```

```

        LTLtemp.add(y.get(0));
        LTLSequence.add(LTLtemp);
    }
    if(total_after.get(act)==0
    &&total_after_resp.get(act)==0){
        LastAct = act;
    } elseif(total_before.get(act)==0
    &&total_before_resp.get(act)==0){
        FirstAct = act;
    }
}

```

**Kode Sumber 4.7. Potongan Declarative Miner Pada Tahap Kedua**

c) Tahap 3

Tahap ketiga dijalankan untuk menentukan nilai 6 variabel, yaitu: totalEXsplit, totalCRsplit, totalCRnext, total EXjoin, totalCRjoin, totalCRbef. Kode Sumber 4.8 – Kode Sumber 4.13 menunjukkan potongan Declarative Miner pada tahap ketiga. Step yang pertama adalah dibuat variabel “y” yang menyimpan aktivitas dari node tujuan, dan variabel “x” yang menyimpan aktivitas dari node asal. Kemudian pada setiap perulangan akan dicari keenam variabel.

c.1. Mencari nilai totalEXsplit

Apabila declaremineroutput.get(k) merupakan exclusive choice maka totalEXsplit bertambah satu.

```

if(declareMinerOutput.getTemplate().get(k).
equals(DeclareTemplate.Exclusive_Choice)){

    totalEXsplit++;

}

```

**Kode Sumber 4.8. Mencari nilai totalEXsplit**

c.2. Mencari nilai totalCRsplit

Apabila declaremineroutput.get(k) merupakan chain response maka totalCRsplit bertambah satu.

```

elseif(declareMinerOutput.getTemplate().get
(k).equals(DeclareTemplate.Chain_Response))
{

    totalCRsplit++;

}

```

**Kode Sumber 4.9. Mencari nilai totalCRsplit**

### c.3. Mencari nilai totalCRnext

Apabila `declaremineroutput.get(k).get(0)` sama dengan `y.get(j)` dan `declaremineroutput.get(k).get(1)` sama dengan aktivitas dan `declaremineroutput.get(k)` merupakan Chain Response maka `totalCRnext` bertambah satu.

```

if(declareMinerOutput.getConstraintParamete
rsMap().get(k).get(0).equals(y.get(j))
&&declareMinerOutput.getConstraintParameter
sMap().get(k).get(1).equals(act)
&&declareMinerOutput.getTemplate().get(k).e
quals(DeclareTemplate.Chain_Response)){

    totalCRnext++;

}

```

**Kode Sumber 4.10. Mencari nilai totalCRnext**

### c.4. Mencari nilai totalEXjoin

Jika diketahui `declaremineroutput.get(k)` merupakan exclusive choice maka `totalEXjoin` bertambah satu.

```

if(declareMinerOutput.getTemplate().get(k).
equals(DeclareTemplate.Exclusive_Choice)){

    totalEXjoin++;

}

```

**Kode Sumber 4.11. Mencari nilai totalEXjoin**

### c.5. Mencari nilai totalCRjoin

Apabila `declaremineroutput.get(k)` merupakan chain response maka `totalCRjoin` bertambah satu.

```
elseif(declareMinerOutput.getTemplate().get(k).equals(DeclareTemplate.Chain_Response))
{
    totalCRjoin++;
}
```

**Kode Sumber 4.12. Mencari nilai totalCRjoin**

#### c.6. Mencari nilai totalCRbef

Jika `declaremineroutput.get(k).get(1)` sama dengan `y.get(j)` dan `declaremineroutput.get(k).get(0)` sama dengan aktivitas dan `declaremineroutput.get(k)` merupakan Chain Response maka `totalCRbef` bertambah satu.

```
if(declareMinerOutput.getConstraintParametersMap().get(k).get(1).equals(x.get(j))
&&declareMinerOutput.getConstraintParametersMap().get(k).get(0).equals(act)
&&declareMinerOutput.getTemplate().get(k).equals(DeclareTemplate.Chain_Response)){
    totalCRbef++;
}
}
```

**Kode Sumber 4.13. Mencari nilai totalCRbef**

#### d) Tahap 4

Tahap keempat merupakan tahapan untuk pembentukan relasi AND, XOR, OR, dan menentukan baik relasi itu Split atau Join. Kode Sumber 4.14 menunjukkan potongan Declarative Miner pada tahap keempat. Jika diketahui suatu aktivitas memiliki `totalEXsplit` sama dengan 0 (nol) maka aktivitas itu berelasi AND split dengan aktivitas berikutnya. Apabila `totalCRnext < total_after` dari suatu

aktivitas maka aktivitas tersebut berelasi XOR split dengan aktivitas berikutnya, jika kedua kondisi tersebut tidak terpenuhi maka aktivitas tersebut berelasi OR split dengan aktivitas berikutnya. Untuk kondisi untuk relasi Join adalah sebagai berikut: Jika suatu aktivitas memiliki totalEXjoin sama dengan 0 (nol) maka bersifat AND join dengan aktivitas sebelumnya. Apabila  $\text{totalCRbef} < \text{total\_before.get(act)}$  maka memiliki relasi XOR join dengan aktivitas sebelumnya. Dan yang terakhir jika dua kondisi join tidak terpenuhi maka aktivitas tersebut memiliki relasi OR join dengan aktivitas sebelumnya.

```
// AND Split
if (total_after.get(act) > 1 && totalCRnext == 0) {
    LTLtemp.add(act);
    if (totalEXsplit == 0) {
        for (int j = 0; j < y.size(); j++) {
            LTLtemp.add(y.get(j));
        }
        LTLOtherSplit.add(LTLtemp);
    }
} else {
    // XOR Split
    if (totalCRnext < total_after.get(act)) {
        for (int j = 0; j < y.size(); j++) {
            LTLtemp.add(y.get(j));
        }
        LTLXORSplit.add(LTLtemp);
    } else {
        // OR Split
        for (int j = 0; j < y.size(); j++) {
            LTLtemp.add(y.get(j));
        }
        LTLOtherSplit2.add(LTLtemp);
    }
}
}
// AND Join
if (LTLSequence.get(k).get(1).equals(act) &&
    LTLSequence.get(k).get(0).equals(x.get(j))) {
```

```

        LTLSequence.remove(k);
        break;
    }
    LTLOtherJoin.add(LTLtemp);
} else {
// XOR Join
if(LTLSequence.get(k).get(1).equals(act) &&
LTLSequence.get(k).get(0).equals(x.get(j))){

    LTLSequence.remove(k);
    break;
}
    LTLXORJoin.add(LTLtemp);
// OR Join
}
else {
    if(LTLSequence.get(k).get(1).equals(act) &&
LTLSequence.get(k).get(0).equals(x.get(j))){

        LTLSequence.remove(k);
        break;
    }
    LTLOtherJoin2.add(LTLtemp);
}
}

```

**Kode Sumber 4.14. Potongan Declarative Miner pada Tahap Keempat**

#### e) Tahap 5

Tahap kelima adalah tahap dimana kita mencari *invisible task*. *Invisible Task* terjadi jika terdapat relasi *XORJoin* dan *XORSplit* secara bersamaan. Sehingga untuk implementasi nya kita perlu mengecek jika ada relasi yang ganda, maka kita perlu untuk menyisipkan aktivitas yang bernama *invisible task* disana

```

for(all_activities)
    int total_appear = 0;
    String act =
declareMinerOutput.getAllActivities().get(i);
    String inv_task = "Invisible_Task";
    for(int j=0;j<LTLXORSplit.size();j++){
        if(LTLXORSplit.get(j).contains(act)){

if(LTLXORSplit.get(j).get(0).equals(act)){
            for(int k=0;k<LTLXORJoin.size();k++){

```

```

        if (LTLXORJoin.get(k).contains(act)) {

if (!LTLXORJoin.get(k).get(0).equals(act)) {
    for (int
l=0;l<LTLXORJoin.get(k).size();l++) {

if (LTLXORJoin.get(k).get(l).equals(act)) {
    LTLtemp.add(inv_task);
    } else {

LTLtemp.add(LTLXORJoin.get(k).get(l));
    }
    }
    LTLXORJoin.remove(k);
    LTLXORJoin.add(LTLtemp);
    break;
    }
    }
    }
}

```

**Kode Sumber 4.15. Potongan Declarative Miner pada Tahap Kelima**

Setelah terbentuk data Linear Temporal Logic (LTL), data harus di proses lebih lanjut agar terbentuk sebuah data eventlog dengan format csv. Hal ini diperlukan dikarenakan untuk mengoutputkan hasil relasi / mengubah eventlog ke dalam link list (bentuk graf Neo4j) diperlukan input dengan format csv (comma delimiter). Perlu diketahui nantinya dalam penelitian kali ini, akan ada pembentukan graf dari 2 eventlog yang berbeda. Eventlog yang pertama yaitu eventlog yang telah diproses terlebih dahulu menggunakan declare miner sehingga semua relasi di dalamnya sudah diketahui. Yang kedua merupakan eventlog yang hanya berisi informasi Case, Activity, dan Waktu (Start Time dan End Time) sehingga agar model bisnis proses yang terbentuk di graf nantinya merupakan suatu process yang benar diperlukan suatu algoritma pada Neo4j untuk mencari relasi tiap activity.

Berikut merupakan algoritma yang akan merubah tiap relasi LTL ke dalam bentuk relasi yang telah ditentukan (LTLXORSPLIT, LTLXORJOIN, LTLSEQUENCE) dengan



format file csv. Kode Sumber 4.16 menjelaskan pengubahan ltl ke format csv pada relasi XOR Split. Program pertama akan membentuk sebuah file baru dengan nama LTLXORSPLIT.csv, kemudian file akan ditambahkan sebuah header yang berisi Case\_ID dan activityname. Berikutnya akan dilakukan *double loop* sebanyak jumlah case XOR Split, dan diambil node asal pada perulangan pertama. Dilanjutkan dengan pengambilan node tujuan pada setiap perulangan tingkat kedua. Program akan berhenti apabila index pada loop telah melebihi jumlah case pada XOR Split.

```

FileWriter fileWriter = new
FileWriter("LTLXORSPLIT.csv");
    fileWriter.append(FILE_HEADER);
    for(int i=0;i<LTLXORSplit.size();i++){
        outputStream.print(LTLXORSplit.get(i).get(0) + " -
><>(");
        for(int j=1;j<LTLXORSplit.get(i).size();j++){

            outputStream.print(LTLXORSplit.get(i).get(j));

            fileWriter.append(NEW_LINE_SEPARATOR);
            fileWriter.append(""+j);
            fileWriter.append(COMMA_DELIMITER);
            fileWriter.append(LTLXORSplit.get(i).get(0));
            fileWriter.append(NEW_LINE_SEPARATOR);
            fileWriter.append(""+j);
            fileWriter.append(COMMA_DELIMITER);
            fileWriter.append(LTLXORSplit.get(i).get(j));
            if(j!=LTLXORSplit.get(i).size()-1){
                outputStream.print(" /\ " );
            }
        }
        outputStream.println("");
    }
fileWriter.close();

```

**Kode Sumber 4.16. Pengubahan LTL ke Format CSV pada Relasi LTLXORSPLIT**

Kode Sumber 4.17 menjelaskan pengubahan ltl ke format csv pada relasi XOR Join. Program pertama akan

membentuk sebuah file baru dengan nama LTLXORJOIN.csv, kemudian file akan ditambahkan sebuah header yang berisi Case\_ID dan activityname. Berikutnya akan dilakukan *double loop* sebanyak jumlah case XOR Join. Program akan mengambil node tujuan pada setiap perulangan tingkat kedua. Pada akhir index perulangan tingkat dua, program akan mengambil node asal, dilanjutkan dengan perulangan pada tingkat satu index berikutnya. Program akan berhenti apabila index pada loop telah melebihi jumlah case pada XOR Join.

```

FileWriter fileWriter = new
FileWriter("LTLXORJOIN.csv");
fileWriter.append(FILE_HEADER);
    for(int i=0;i<LTLXORJoin.size();i++){
        outputStream.print("<>");
        for(int
j=1;j<LTLXORJoin.get(i).size();j++){

            outputStream.print(LTLXORJoin.get(i).get(j));

            fileWriter.append(NEW_LINE_SEPARATOR);
            fileWriter.append(""+j);
            fileWriter.append(COMMA_DELIMITER);
            fileWriter.append(LTLXORJoin.get(i).get(j));
            fileWriter.append(NEW_LINE_SEPARATOR);
            fileWriter.append(""+j);
            fileWriter.append(COMMA_DELIMITER);
            fileWriter.append(LTLXORJoin.get(i).get(0));
                if(j!=LTLXORJoin.get(i).size()-
1){
                    outputStream.print(" /\\"
");
                }
            }

            outputStream.println(" ->
O("+LTLXORJoin.get(i).get(0)+ " " );
        }
fileWriter.close();

```

**Kode Sumber 4.17. Pseudocode Pengubahan LTL ke Format CSV pada Relasi LTLXORJOIN**

Kode Sumber 4.18 menjelaskan pengubahan ltl ke format csv pada relasi Sequence. Program pertama akan membentuk sebuah file baru dengan nama LTLSequence.csv, kemudian file akan ditambahkan sebuah header yang berisi Case\_ID dan activityname. Berikutnya akan dilakukan *double loop* sebanyak jumlah case LTL Sequence, dan diambil node asal pada perulangan pertama. Dilanjutkan dengan pengambilan node tujuan pada perulangan tingkat kedua. Program akan berhenti apabila index pada loop telah melebihi jumlah case pada LTL Sequence.

```

FileWriter fileWriter = new
FileWriter("LTLSequence.csv");
fileWriter.append(FILE_HEADER);
outputStream.println("LTLSequence");
    for(int i=0;i<LTLSequence.size();i++){
        outputStream.println(LTLSequence.get(i)
.get(0) + " -> O(" + LTLSequence.get(i).get(1)+
" )");
        for(int
j=0;j<LTLSequence.get(i).size();j++){
            fileWriter.append(NEW_LINE_SEPARATOR);
            fileWriter.append(""+i+1);
            fileWriter.append(COMMA_DELIMITER)
            fileWriter.append(LTLSequence.get(i).get(j));
        }
    }
fileWriter.close();

```

**Kode Sumber 4.18. Pseudocode Pengubahan LTL ke Format CSV pada Relasi LTLSequence**

Setelah terbentuk file dengan format csv, gunakan Kode Sumber 4.19 untuk membentuk tiap file ke bentuk link list ( node, dan juga relasinya ). File LTLXORSPLIT.csv akan diubah ke relasi XOR Split pada link list, file LTLXORJOIN.csv akan diubah ke relasi XOR Join pada link list, file LTLSequence.csv akan diubah ke relasi NEXT pada link list. Program akan mengambil baris genap pada tiap file untuk dijadikan node awal, kemudian

node ganjil pada tiap file akan dijadikan node tujuan. Relasi yang menghubungkan node awal dan node tujuan akan bergantung pada nama file yang digunakan.

```
//for detecting XORSplit
LOAD CSV WITH HEADERS FROM "file XORSPLIT" AS line
MERGE (n:load { Case_ID:line.Case_ID, Activity:line.activityname }) WITH COLLECT(n) AS activities
FOREACH (m IN RANGE(0, SIZE(activities)/2-1) |
  FOREACH (prev IN [activities[m*2]] |
    FOREACH (next IN [activities[m*2+1]] |
      MERGE (prev)-[:NEXT]->(next) MERGE (prev)-[:XORSPLIT]->(next)))
//for detecting XORJoin
LOAD CSV WITH HEADERS FROM "file XORJOIN" AS line
MERGE (n:load { Case_ID:line.Case_ID, Activity:line.activityname }) WITH COLLECT(n) AS activities
FOREACH (m IN RANGE(0, SIZE(activities)/2-1) |
  FOREACH (prev IN [activities[m*2]] |
    FOREACH (next IN [activities[m*2+1]] |
      MERGE (prev)-[:NEXT]->(next) MERGE (prev)-[:XORJOIN]->(next)))
//for detecting Sequence
LOAD CSV WITH HEADERS FROM "file:///sequence.csv" AS line
MERGE (n:load { Case_ID:line.Case_ID, Activity:line.activityname }) WITH COLLECT(n) AS activities
FOREACH (m IN RANGE(0, SIZE(activities)/2-1) |
  FOREACH (prev IN [activities[m*2]] |
    FOREACH (next IN [activities[m*2+1]] |
      MERGE (prev)-[:NEXT]->(next)))
```

#### **Kode Sumber 4.19. Memodelkan Model Bisnis Proses dari LTL**

### **4.2.1.2.2 Implementasi Pembentukan Business Process Model dari Model Neo4J**

Untuk mengimplementasikan pembentukan Business Process model dari model Neo4J, langkah pertama kita harus membuat link list dari event log yang berbentuk csv. Untuk kode sumber nya ada di Kode Sumber 4.20 .

```
//LOAD CSV
LOAD CSV with headers FROM "file e" AS line
Merge (:Activity
{CaseID:line.Case_ID,Name:line.Activity,StartTime:line.start_stamp,EndTime:line.End_stamp })
LOAD CSV with headers FROM "file:///eventlogswitch.csv" AS line
Merge (:CaseActivity {Name:line.Activity })
//CONNECT THE ACTIVITY
Match (c:Activity)
WITH COLLECT(c) AS caselist
UNWIND RANGE(0,size(caselist) - 2) as idx
WITH caselist[idx] AS s1, caselist[idx+1] AS s2
match (b:CaseActivity),(a:CaseActivity)
WHERE s1.CaseId = s2.CaseId AND s1.Name = a.Name AND s2.Name = b.Name
MERGE (a)-[r:NEXT]->(b)
```

#### **Kode Sumber 4.20. Mengubah event log ke link list**

Setelah berhasil mendapatkan link list, maka kita akan mendeteksi relasi antar aktivitas dengan melihat outgoing relation dan incoming relation seperti pada Kode Sumber 4.21 .

```
match (n)-[r:NEXT]->(a)
where size((n)-->())>1 and (size((a)<--()) = 1 and size((a)-->()) = 1)
Merge(n)-[:XORSplit]->(a)
Return distinct 'XORSplit', n.Name, a.Name|

match (n)-[r:NEXT]->(a)
where size((n)-->())>1 and (size((a)<--()) > 1)
Merge(n)-[:XORJoin]->(a)
Return distinct 'XORJoin', n.Name, a.Name
```

#### **Kode Sumber 4.21. Membentuk Relasi di Neo4J**

Setelah mengetahui relasi maka kita mengecek invisible task dan jika ada kita memperbaiki graph model tersebut seperti pada Kode Sumber 4.22.

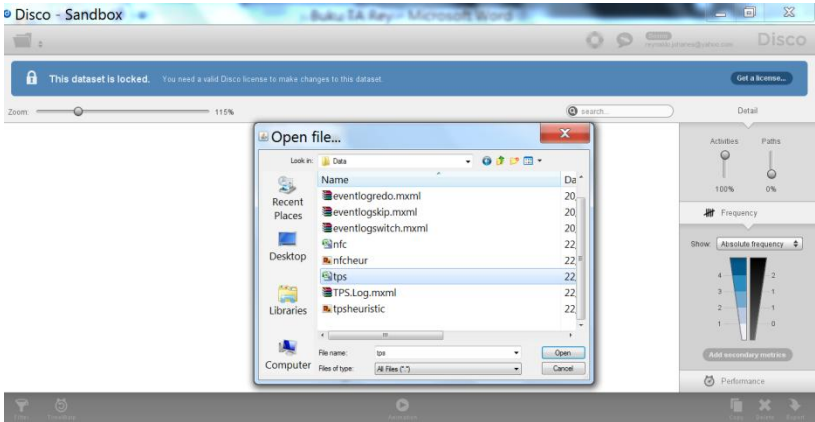
```
match (n)-[r:NEXT]->(a)
where size((n)-->())>1 and (size((a)<--()) > 1 and size((a)-->()) > 1)
create (i:invisibletask)
create (n)-[:NEXT]->(i)
create (i)-[:NEXT]->(a)

match (n)-[r:NEXT]->(a)
where size((n)-->())>1 and (size((a)<--()) > 1 )
create (i:invisibletask)
create (n)-[:NEXT]->(i)
create (i)-[:NEXT]->(a)
```

#### **Kode Sumber 4.22. Mengecek dan Memperbaiki Invisible Task**

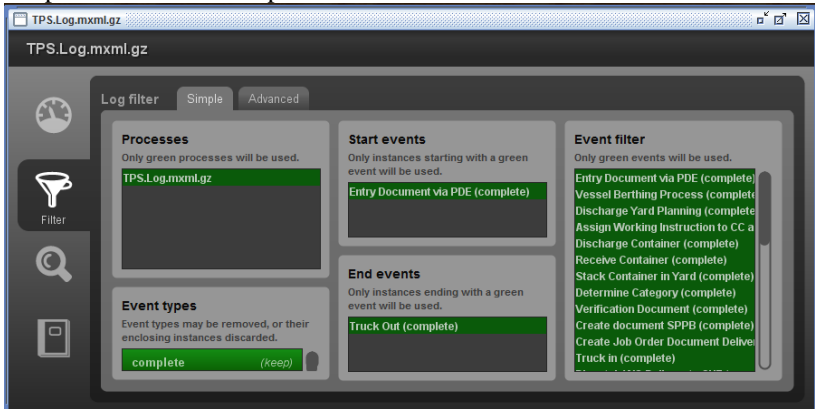
### **4.2.1.2.3 Implementasi Pembentukan Business Process Model dari Model Heuristic Miner**

Untuk memodelkan pembentukan Business Process Model dari Model Heuristic Miner maka langkah pertama yang harus kita lakukan adalah membuat ekstensi mxml dari event log. Buka Disco serta masukkan event log ke dalam disco serta export ke mxml dari disco seperti di Gambar 4.1.

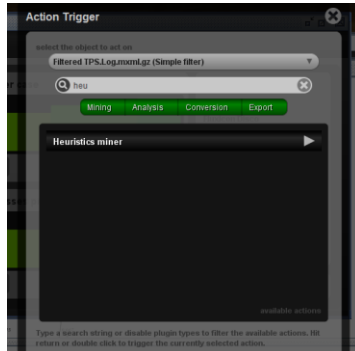


**Gambar 4.1. Memasukkan Event Log ke Disco**

Setelah mendapatkan ekstensi mxml kita masukkan ke dalam ProM seperti pada Gambar 4.2. Lalu kita pilih start analyzing this log dan pilih heuristic miner pada Gambar 4.3.

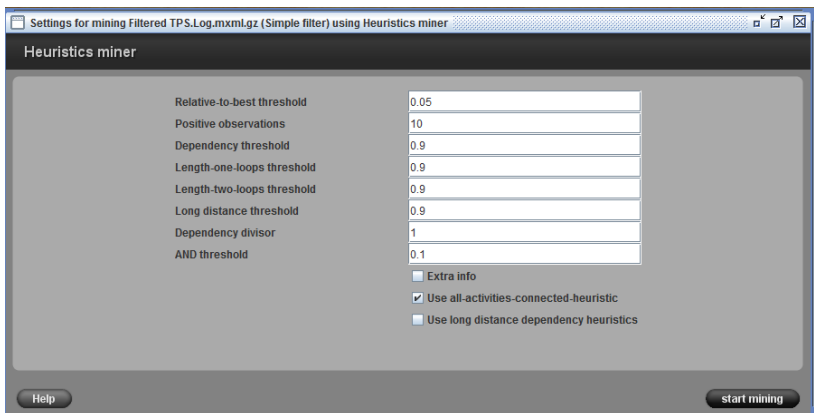


**Gambar 4.2. Memasukkan Event Log ke ProM**



**Gambar 4.3. Heuristic Miner**

Setelah itu kita harus mengisi nilai dari Relative to best threshold, Positive observations, Dependency threshold, Length one loops threshold, length two loops threshold, long distance threshold, Dependency divisor, dan And threshold, tapi kita akan memakai nilai default semua. Setelah semua nilai terisi, maka kita melakukan start analyzing seperti pada Gambar 4.4.



**Gambar 4.4. Memasukkan nilai untuk melakukan Heuristic Miner**

*[Halaman ini sengaja dikosongkan]*



## **BAB V**

### **PENGUJIAN DAN EVALUASI**

Bab ini membahas hasil dan pembahasan pada aplikasi yang dikembangkan. Pada bab ini akan dijelaskan tentang data yang digunakan, hasil yang didapatkan dari penggunaan perangkat lunak dan uji coba yang dilakukan pada perangkat lunak yang telah dikerjakan untuk menguji apakah fungsionalitas aplikasi telah diimplementasikan dengan benar dan berjalan sebagaimana mestinya.

#### **5.1 Lingkungan Uji Coba**

Lingkungan uji coba menjelaskan lingkungan yang digunakan untuk menguji implementasi pembuatan sistem pada tugas akhir ini. Lingkungan uji coba meliputi perangkat keras dan perangkat lunak yang dijelaskan sebagai berikut:

1. Perangkat keras
  - a. Prosesor : AMD A8-5545M APU @ 1.70GHz.
  - b. Memori(RAM) : 8 GB.
  - c. Tipe sistem : 64-bit sistem operasi.
2. Perangkat lunak
  - a. Sistem operasi : Windows 8.1 Single Language.
  - b. Kakas bantu : Java, Neo4J, ProM, Eclipse, SQL.

#### **5.2 Pre-Processing**

##### **5.2.1 Hasil Transformasi ke Log Data**

Database TPS yang digunakan dapat dilihat pada Gambar 5.1. Contoh kolom yang diolah bukan sebagai waktu dari aktivitas atau *message* adalah kolom Container\_Key dan Container\_Type. Kolom paling kiri adalah kolom Container\_Key yang digunakan sebagai Case\_ID dan kolom Container\_Type akan digunakan sebagai atribut yang disebut sebagai detail lampiran pada log data. Kemudian, log data tersebut diolah menggunakan metode yang dijelaskan pada Sub

Bab 3.3.1. Hasil dari pengolahan log data dapat dilihat pada Gambar 5.2. Pada log data yang tergambar di Gambar 5.2, *message* maupun aktivitas masih terekam menjadi satu di kolom nama aktivitas.

## 5.2.2 Hasil Pemisahan Message dari Log Data serta Pengelompokan Data per Bulan

Agar dapat digunakan untuk pemodelan, log data di-*filter* terlebih dahulu dengan memisahkan antara *message* dan aktivitas. **Gambar 5.3** menunjukkan log data yang sudah di-*filter* dari *message*. Jika dilihat pada Log yang sudah di-*filter*, terdapat beberapa aktivitas seperti *Request Behandle* dan *Approve Behandle* terhapus dari log. Itu menunjukkan bahwa aktivitas-aktivitas tersebut merupakan *message*. Kemudian, log yang sudah di-*filter* tersebut akan dikelompokkan menjadi Log Data Bulan Januari, Log Data Bulan Februari, dan Log Data Bulan Maret sebagai bahan uji coba pembentukan model dan perbaikan pada proses yang terpotong.

CONTA	CTR_SI	CTR_TY	GROSS	VESSEL_ATB	DISC_DATE	YARD	YARD	STACK_DATE
4484646	40	DRY	29.103	1/31/16 14:35	2/1/16 20:22	M	112	2/1/16 21:00
4484654	40	DRY	28.408	1/31/16 14:35	2/1/16 20:24	I	126	2/1/16 21:14
4485330	20	DRY	24.82	1/31/16 14:35	2/1/16 21:52	S	13	2/1/16 22:13
4484680	20	DRY	22.235	1/31/16 14:35	2/1/16 21:54	S	13	2/1/16 22:17
4484670	20	DRY	23.066	1/31/16 14:35	2/1/16 21:55	S	13	2/1/16 22:18
4484683	20	DRY	23.085	1/31/16 14:35	2/1/16 22:04	S	13	2/1/16 22:22
4484679	20	DRY	27.1	1/31/16 14:35	2/1/16 22:06	S	13	2/1/16 22:24
4484669	20	DRY	22.707	1/31/16 14:35	2/1/16 22:08	S	29	2/1/16 22:26
4484663	20	DRY	13.608	1/31/16 14:35	2/1/16 22:11	S	29	2/1/16 22:27
4484862	20	DRY	29.245	1/31/16 14:35	2/1/16 1:08	M	27	2/1/16 1:22
4485109	40	DRY	23.29	1/31/16 14:35	2/1/16 1:09	M	30	2/1/16 1:30
4484857	20	DRY	23.088	1/31/16 14:35	2/1/16 1:10	M	27	2/1/16 1:23

**Gambar 5.1**Database Proses Impor Barang Terminal Peti Kemas

Case ID	Sender	Original	Input	Activity	Output	Receiver	Time	Cost	Yard	Yard	Detail Lampiran
4453421		Customer	NPWP, SI	Document_Entry_via_PDE	BC 2.0		23/01/2016 7:22	0			Dry;Green Line
4453421	Customer		BC 2.0	Request_Behandle	BC 2.0	SKP	23/01/2016 7:23	0			
4453421		TPS		Vessel_Berthing_Process			23/01/2016 10:10	47836,7172			
4453421		TPS		Discharge_Container			23/01/2016 20:32	428,1331788			
4453421		TPS		Bring_Container_to_Yard			23/01/2016 20:51	50			
4453421		TPS		Stack_Container_in_Yard			23/01/2016 20:54	19,34	J	61	
4453421	SKP		BC 2.0	Approve_Behandle	BC 2.0	Customer	24/01/2016 11:07	1,467092317			
4453421		SKP	BC 2.0	Verification_Document_Beh	BC 2.0		25/01/2016 0:55	6,210446525			Dry;Green Line
4453421		Pejabat B/LHP, BC 2.		Create_document_SPPB	SPPB		25/01/2016 7:48	3450,546591			
4453421	Pejabat Bea Cukai		SPPB	Send_SPPB_Info		Customer	25/01/2016 7:48	0			
4453421		Customer	SPPB	Create_Job_Order_Documen	CEIR		25/01/2016 15:33	45,79705642			
4453421	Customer			Send_Job_Order_Delivery_Info		TPS	25/01/2016 15:35	0			
4453421		Customer		Truck_in			27/01/2016 15:34	11,00157762			
4453421		TPS		Dispatch_WQ_Delivery_to_CHE			27/01/2016 15:35	1,330475647			
4453421		TPS		Determine_Container_Type			27/01/2016 15:38	1,810399109			
4453421		TPS		Determining_Dry			27/01/2016 15:40	14,79			

**Gambar 5.2 Log Data Proses Impor Barang Terminal Peti Kemas**

Case ID	Sender	Original	Input	Activity	Output	Receiver	Time	Cost	Yard	Yard	Detail Lampiran
4453421		Customer	NPWP, SI	Document_Entry_via_PDE	BC 2.0		23/01/2016 7:22	0			Dry;Green Line
4453421		TPS		Vessel_Berthing_Process			23/01/2016 10:10	47836,72			
4453421		TPS		Discharge_Container			23/01/2016 20:32	428,1332			
4453421		TPS		Bring_Container_to_Yard			23/01/2016 20:51	50			
4453421		TPS		Stack_Container_in_Yard			23/01/2016 20:54	19,34	J	61	
4453421	SKP		BC 2.0	Verification_Document_Behandle	BC 2.0		25/01/2016 0:55	6,210447			Dry;Green Line
4453421	Pejabat B/LHP, BC 2.			Create_document_SPPB	SPPB		25/01/2016 7:48	3450,547			
4453421	Customer		SPPB	Create_Job_Order_Document_Deli	CEIR		25/01/2016 15:33	45,79706			
4453421		Customer		Truck_in			27/01/2016 15:34	11,00158			
4453421		TPS		Dispatch_WQ_Delivery_to_CHE			27/01/2016 15:35	1,330476			
4453421		TPS		Determine_Container_Type			27/01/2016 15:38	1,810399			
4453421		TPS		Determining_Dry			27/01/2016 15:40	14,79			
4453421		TPS		Decide_Task_Before_Lift_Container			27/01/2016 15:42	3,166661			
4453421		TPS		Lift_on_Container_Truck			27/01/2016 15:44	15,98			
4453421		Customer		Truck_Go_To_Gate_Out			27/01/2016 16:30	1,992126			

**Gambar 5.3 Log Data Tanpa Message pada Proses Impor Barang Terminal Peti Kemas**

## 5.3 Hasil Implementasi Pembentukan Business Process Model

Di bab ini akan dijelaskan mengenai hasil implementasi Pembentukan Business Process Model. Hasil ini akan dibagi 3 yaitu hasil implementasi pembentukan business process model dari model deklaratif, Neo4J, dan Heuristic Miner.

### 5.3.1 Hasil Implementasi Pembentukan Business Process Model dari Model Deklaratif

Didapatkan hasil implementasi dari metode yang dijalankan . Terdapat 5 jenis LTL disini yaitu LTLFirstLast untuk mendapatkan aktivitas pertama dan terakhir, LTL Sequence untuk mendapatkan aktivitas yang berurutan, LTL XorSPLIT untuk mendapatkan aktivitas yang mempunyai relasi xor split, LTL XorJOIN untuk mendapatkan aktivitas yang mempunyai relasi xor join, LTL NonFreeChoice untuk mendapatkan aktivitas yang mempunyai relasi non free choice. Isi dari setiap jenis LTL bisa dilihat dari Tabel 5.1 . Di sini merupakan hasil uji coba dari Skip Decision.

**Tabel 5.1. Hasil LTL TPS**

Jenis LTL	Isi LTL
LTLFirstLast	Firstactivity(Entry Document via PDE-complete) Lastactivity(Truck Out-complete)
LTLSequence	Discharge Yard Planning-complete -> O(Assign Working Instruction to CC and Yard-complete) Entry Document via PDE-complete -> O(Vessel Berthing Process-complete) Bring Container to Behandle Area-complete -> O(Check Goods Behandle-complete) Create Job Order Document Delivery-complete -> O(Truck in-complete) Truck Go To Gate Out-complete -> O(Check Container before Truck out-complete) Receive Container-complete -> O(Stack Container in Yard-complete) Assign Working Instruction to CC and Yard-complete -> O(Discharge Container-complete)

Jenis LTL	Isi LTL
	<p>Lift on Container Truck-complete -&gt;  O(Truck Go To Gate Out-complete)  Check Goods Behandle-complete -&gt;  O(Bring Back To Yard from Behandle-complete)  Bring Back To Yard from Behandle-complete -&gt; O(Create SPPB-complete)  Vessel Berthing Process-complete -&gt;  O(Discharge Yard Planning-complete)  Create Job Order Document Behandle-complete -&gt; O(Bring Container to Behandle Area-complete)  Truck in-complete -&gt; O(Dispatch WQ Delivery to CHE-complete)  Verification Document-complete -&gt;  O(Create document SPPB-complete)  Discharge Container-complete -&gt;  O(Receive Container-complete)  Create Job Order Document Quarantine-complete -&gt; O(Bring Container to Quarantine Area-complete)  Check Goods Quarantine-complete -&gt;  O(Bring Back To Yard from Quarantine-complete)  Bring Container to Quarantine Area-complete -&gt; O(Check Goods Quarantine-complete)  Check Container before Truck out-complete -&gt; O(Truck Out-complete)  Dispatch WQ Delivery to CHE-complete -&gt; O(Determine Container Type-complete)</p>
LTLXORSplit	<p>Determine Category-complete -&gt;  O(Create Job Order Document Behandle-complete) ∨ Verification Document-</p>

Jenis LTL	Isi LTL
	complete) Determine Container Type-complete -> O(Determining Reefer-complete ∨ Determining Dry-complete ∨ Determining Uncontainer-complete) Stack Container in Yard-complete -> O(Invisible_Task ∨ Create Job Order Document Quarantine-complete)
LTLXORJoin	O(Create SPPB-complete ∨ Create document SPPB-complete) -> O(Create Job Order Document Delivery-complete) O(Bring Back To Yard from Quarantine- complete ∨ Invisible_Task) -> O(Determine Category-complete) O(Prepare Tools-complete ∨ Invisible_Task ∨ Unplug Reefer- complete) -> O(Lift on Container Truck- complete)
LTLNonFreeChoice	$\langle \rangle ((\text{Determining Uncontainer-complete} \wedge \text{Decide Task Before Lift Container-complete} \wedge \text{Prepare Tools-complete}) \vee (\text{Determining Reefer-complete} \wedge \text{Decide Task Before Lift Container-complete} \wedge \text{Unplug Reefer-complete}) \vee (\text{Invisible\_Task} \wedge \text{Decide Task Before Lift Container-complete} \wedge \text{Determining Dry-complete}))$

Setelah mendapatkan LTL itu kemudian diubah ke dalam cypher dan hasilnya dapat di lihat di Gambar 5.4. Node yang berwarna merah ialah invisible task. Gambar utuh hasil pemodelan dapat dilihat di Lampiran.

Karena TPS hanya mengandung skip decision maka kita akan mencoba untuk kasus yang mengulang dengan *trace*: [ABCF, ADEF, ABEF ]dan berpindah dengan *trace* : [ABCBCD, ABCBCBCD, ABCBCD]. Hasil LTL yang bisa didapatkan pada *trace* tersebut ialah pada Tabel 5.2. dan Tabel 5.3.Di sini merupakan hasil uji coba dari Redo dan Switch.

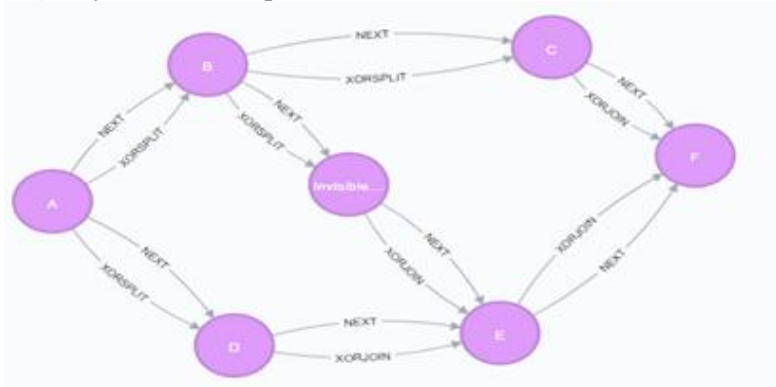
Table	LTL List
LTLFirstLast	$Firstactivity(A)$
	$Lastactivity(F)$
LTLXorSplit	$A \rightarrow O( ( BVD ) )$
	$B \rightarrow O( ( CVInvisible \ Task$ $) )$

LT LXorJoin	$O((DVInvisible\_Task)) - > O(E)$
	$O((C V E)) -> O(F)$

Tabel 5.3. Hasil LTL Redo

Table	LTL List
LTLFirstLast	<i>Firstactivity</i> (A)
	<i>Lastactivity</i> (F)
LTLXorSplit	$A -> O((BVD))$
	$B -> O((CVInvisible\ Task))$
LT LXorJoin	$O((DVInvisible\_Task)) - > O(E)$
	$O((C V E)) -> O(F)$

Setelah didapatkan hasil LTL maka didapatkan juga bentuk *graph* nya di *Neo4J* seperti di Gambar 5.5 dan Gambar 5.6



Gambar 5.5. Hasil Switch dari LTL

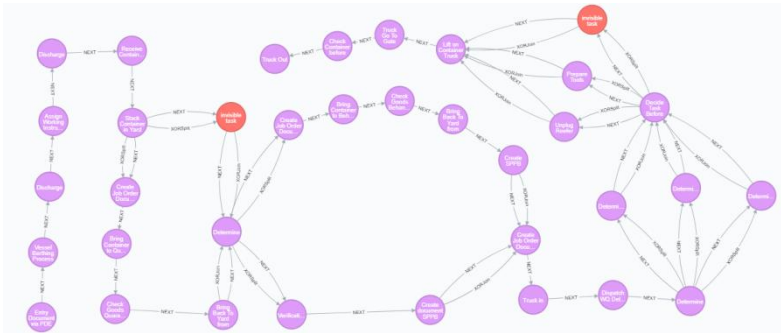


### 5.3.2 Hasil Implementasi Pembentukan Business Process Model dari Model Neo4J

[illegible]

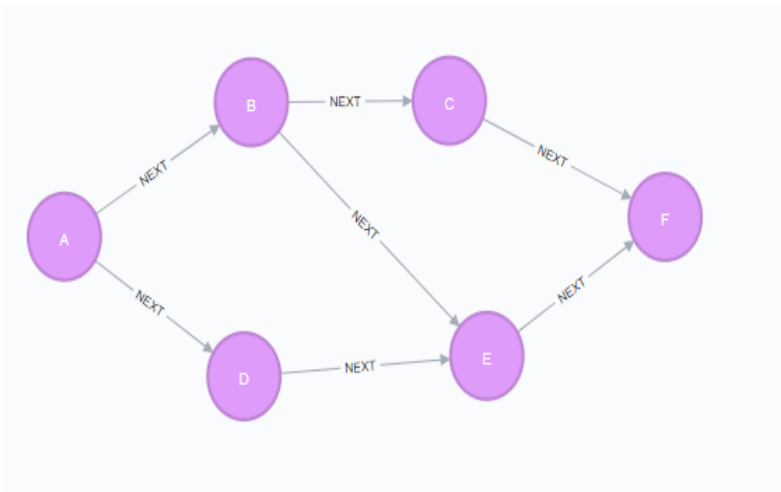
### Gambar 5.7. Hasil Link List

Gambar 5.8 menunjukkan Business Process Model dalam bentuk graph. Node yang berwarna merah ialah invisible task.

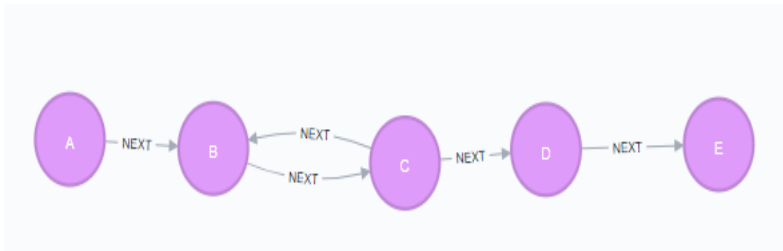


**Gambar 5.8. Hasil Graph dari Neo4J**

Karena TPS hanya mengandung skip decision maka kita akan mencoba untuk kasus yang mengulang dengan *trace*: [ ABCF, ADEF, ABEF ] dan berpindah dengan *trace* : [ ABCBCD, ABCBCBCD, ABCBCD ]. Hasil Link List yang bisa didapatkan pada *trace* tersebut ialah pada Gambar 5.9. dan Gambar 5.10.

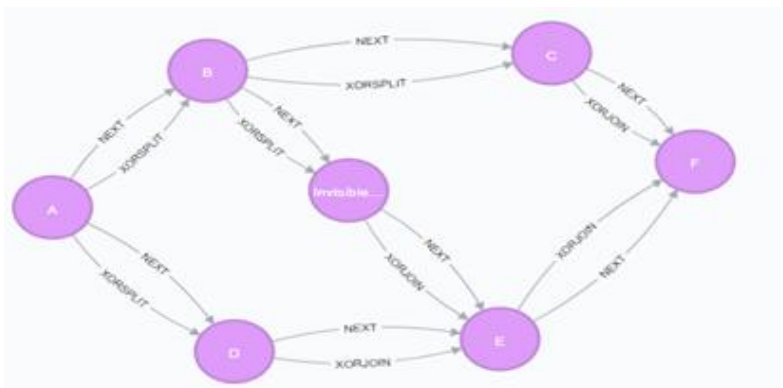


**Gambar 5.9. Hasil Link List Switch**

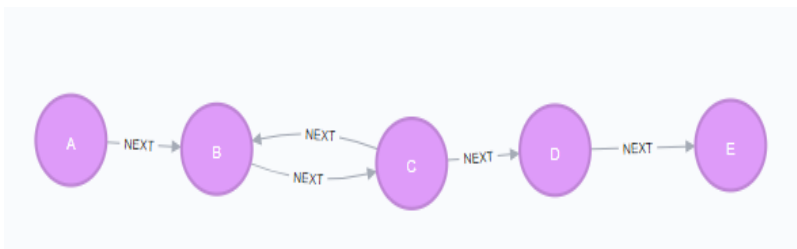


**Gambar 5.10. Hasil Link List Redo**

Setelah mendapatkan *Link List*, kita akan memberi relasi dan menyisipkan *invisible task* jika ada. Hasil dapat dilihat di Gambar 5.11 dan Gambar 5.12.



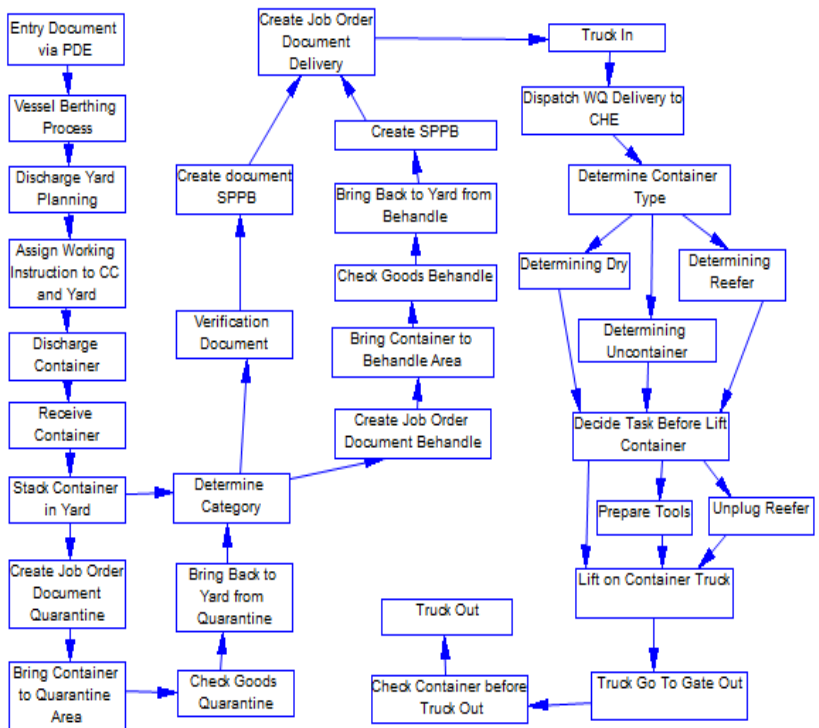
**Gambar 5.11. Hasil Switch langsung dari Event Log**



**Gambar 5.12. Hasil Redo langsung dari Event Log**

### 5.3.3 Hasil Implementasi Pembentukan Business Process Model dari Model Heuristic Miner

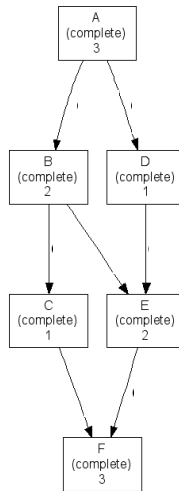
Setelah melakukan metode untuk melakukan pembentukan Business Process Model dari Model Heuristic Miner, maka akan keluar hasil seperti pada Gambar 5.13 .



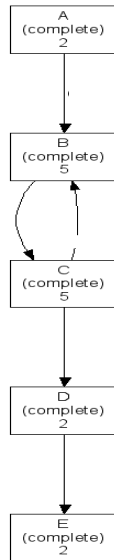
**Gambar 5.13. Hasil Graph dari Heuristic Miner**

Karena TPS hanya mengandung skip decision maka kita akan mencoba untuk kasus yang mengulang dengan *trace*: [

ABCF, ADEF, ABEF ]dan berpindah dengan *trace* : [ABCBCD, ABCBCBCD, ABCBCD]. Hasil LTL yang bisa didapatkan pada *trace* tersebut ialah pada Gambar 5.14 dan Gambar 5.15.



**Gambar 5.14. Hasil Switch dari Heuristic Miner**



**Gambar 5.15. Hasil Redo dari Heuristic Miner**

## 5.4 Hasil Perbandingan 3 Graph

Dari Ketiga hasil Graph yang telah dibentuk oleh *Declarative Miner*, *Neo4J*, dan *Heuristic Miner* kita dapat mengetahui hasil terbaik dimana hasil terbaiknya adalah hasil Neo4J dan hasil LTL. Hal ini karena penambang heuristik tidak bisa langsung menghubungkan relasi antar aktivitas. Sedangkan untuk mendapatkan relasi kita harus konversi terlebih dahulu ke petri net. Meski kita mendapatkan relasi antar aktivitas, relasinya masih salah jika event log mengandung *invisible task*. Misalnya hubungan antara A dan C pada Gambar . Hubungan antara A dan C adalah keduanya XORJoin atau XORSplit. Satu node memiliki dua relasi sehingga relasinya salah. Sebaliknya, hasil Neo4J bisa langsung menghasilkan relasi yang benar padahal log kejadian mengandung *invisible task*.

## **BAB VI**

### **KESIMPULAN DAN SARAN**

Pada bab ini akan diberikan kesimpulan yang diambil selama pengerjaan Tugas Akhir serta saran-saran tentang pengembangan yang dapat dilakukan terhadap Tugas Akhir ini di masa yang akan datang.

#### **6.1 Kesimpulan**

Dari hasil pengamatan selama proses perancangan, implementasi, dan pengujian perangkat lunak yang dilakukan, dapat diambil kesimpulan sebagai berikut:

1. Data dari database ditransform dengan baik dan benar menjadi event log yang dapat diolah.
2. Pembentukan rule *LTL* terhadap log data yang mengandung *invisible task* dapat dilakukan dengan menggunakan *declarative miner*
3. Pembentukan *graph model* dari *LTL* dapat dilakukan dengan menggunakan Neo4J.
4. *Graph Model* yang dibentuk dari event log dengan menggunakan Neo4J dapat mengatasi *invisible task*.
5. *Graph Model* yang dibentuk dari event log dengan menggunakan *Heuristic Miner* tidak dapat mengatasi *invisible task*.
6. Metode menggunakan Neo4J dan *Declarative Miner* merupakan metode yang paling tepat untuk menggambarkan *Business Process Model*.

#### **6.2 Saran**

Berikut merupakan saran untuk pengembangan sistem di masa yang akan datang. Saran ini didasarkan pada hasil perancangan, implementasi dan pengujian yang telah dilakukan. Saran tersebut adalah perbaikan pada algoritma *Heuristic Miner* agar dapat menangani event log yang mengandung *Invisible Task*.

*[Halaman ini sengaja dikosongkan]*



## DAFTAR PUSTAKA

- [1] L. Wen, J. Wang, W. M. P. van der Aalst, B. Huang, and J. Sun, "Mining process models with prime invisible tasks," *Data & Knowledge Engineering*, vol. 69, no. 10, pp. 999–1021, Oct. 2010.<https://doi.org/10.1016%2Fj.datak.2010.06.001>
- [2] R. Sarno, Y. A. Effendi, and F. Haryadita, "Modified Time-Based Heuristics Miner for Parallel Business Processes," *Int. Rev. Comput. Softw.*, vol. 11, no. 3, pp. 249–260, 2016.<https://doi.org/10.15866/irecos.v11i3.8717>
- [3] N. Y. Setiawan and R. Sarno, "Multi-Criteria Decision Making for Selecting Semantic Web Service Considering Variability and Complexity Trade-Off," *J. Theor. Appl. Inf. Technol.*, vol. 86, no. 2, pp. 316–326, 2016.
- [4] S. R.a *et al.*, "Developing a workflow management system for enterprise resource planning," *IAENG Int. J. Comput. Sci.*, vol. 72, no. 3, pp. 412–421, 2015.
- [5] R. Sarno and K. R. Sungkono, "Coupled Hidden Markov Model for Process Mining of Invisible Prime Tasks," *Int. Rev. Comput. Softw.*, vol. 11, no. 6, pp. 539–547, 2016.<https://doi.org/10.15866/irecos.v11i6.9555>
- [6] R. Sarno and K. R. Sungkono, "Hidden Markov Model for Process Mining of Parallel Business Processes," *Int. Rev. Comput. Softw.*, vol. 11, no. 4, pp. 290–300, 2016.<https://doi.org/10.15866/irecos.v11i4.8700>
- [7] S. Huda, R. Sarno, and T. Ahmad, "Increasing accuracy of process-based fraud detection using a behavior model," *Int. J. Softw. Eng. its Appl.*, vol. 10, no. 5, pp. 175–188, 2016.<https://doi.org/10.14257/ijseia.2016.10.5.16>
- [8] W. Chomyat and W. Premchaiswadi, "Process mining on medical treatment history using conformance checking," 2016.<https://doi.org/10.1109/ictke.2016.7804102>
- [9] T. Erdogan and A. Tarhan, "Process Mining for Healthcare Process Analytics," 2016.<https://doi.org/10.1109/iwsm-mensura.2016.027>

- [10] A. S. Osses, "Business process analysis in advertising: An extension to a methodology based on process mining projects," 2016.<https://doi.org/10.1109/sccc.2016.7836000>
- [11] W.M.P. Van Der Aalst, "Process Mining: Discovery, Conformance and Enhancement of Business Processes," 2011.

## LAMPIRAN

1. Hasil *Graph Model* yang dimodelkan *LTL*
2. Hasil *Graph Model* yang dimodelkan *Neo4J*
3. List Aktivitas yang terdapat didalam *Database TPS*
4. *SourceCode* untuk membentuk *Control-Flow Patterns* menggunakan *Declarative Miner*

*[Halaman ini sengaja dikosongkan]*

## DAFTAR ISTILAH

<i>AND</i>	: relasi yang menunjukkan bahwa seluruh aktivitas pada relasi ini harus dijalankan semua di tiap proses.
<i>Case</i>	: proses
<i>Fitness</i>	: nilai kecocokan
<i>Invisible Task</i>	: aktivitas tidak terlihat (aktivitas yang tidak terdapat dalam log data, tetapi dibutuhkan di model proses )
<i>Invisible Prime Task</i>	: aktivitas utama tak terlihat
<i>Invisible Non-Prime Task</i>	: aktivitas bukan utama tak terlihat
<i>Non-free choice</i>	: pilihan tidak bebas
<i>OR</i>	: relasi yang menunjukkan bahwa seluruh aktivitas pada relasi tersebut dapat dijalankan semua atau dipilih beberapa untuk tiap proses
<i>Node</i>	: unit dalam graf
<i>Precision</i>	: nilai presisi
<i>Process Discovery</i>	: pembentukan model proses
<i>Process Mining</i>	: penggalian proses
<i>Skip activity</i>	: aktivitas yang hilang
<i>SOP</i>	: <i>Standard Operating Procedure</i>
<i>Trace</i>	: varian proses pada log data
<i>Truncated Process</i>	: proses yang terpotong
<i>XOR</i>	: relasi yang menunjukkan bahwa seluruh aktivitas pada relasi ini harus dipilih salah satu untuk dijalankan
<i>YAWL</i>	: <i>Yet Another Workflow Language</i>

*[Halaman ini sengaja dikosongkan]*

## BIODATA PENULIS



Reynaldo Johanes, lahir pada tanggal 9 Oktober 1996 di Surabaya. Penulis merupakan seorang mahasiswa yang sedang menempuh studi di Jurusan Teknik Informatika Institut Teknologi Sepuluh Nopember (ITS). Memiliki beberapa hobi antara lain berolahraga dan bermain. Pernah menjadi asisten dosen di PIKTI ITS. Selama menempuh pendidikan di kampus, penulis juga aktif dalam organisasi kemahasiswaan, antara lain Staff Departemen Kewirausahaan Himpunan Mahasiswa Teknik Computer-Informatika (HMTIC), Staff Schematics, panitia ITS Futsal Championship pada tahun ke-2, serta Staff Ahli Teknis National Logic Competition pada tahun ke-3.

*[Halaman ini sengaja dikosongkan]*